

**DTIC FILE COPY**

**AD-A222 760**

**AAMRL-TR-88-041**



**THE RETRIEVAL OF  
INFORMATION FROM SECONDARY  
MEMORY: A REVIEW AND NEW  
FINDINGS (U)**

**David L. Strayer  
Arthur F. Kramer**

**UNIVERSITY OF ILLINOIS**

**DECEMBER 1989**

**FINAL REPORT FOR PERIOD SEPTEMBER 1987 - MARCH 1988**

**DTIC  
ELECTE  
JUN 04 1990**  
**S E D**  
*Co*

**Approved for public release; distribution is unlimited.**

**HARRY G. ARMSTRONG AEROSPACE MEDICAL RESEARCH LABORATORY  
HUMAN SYSTEMS DIVISION  
AIR FORCE SYSTEMS COMMAND  
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6573**

**93 05 81 226**

## NOTICES

When US Government drawings, specifications, or other data are used for any purpose other than a definitely related Government procurement operation, the Government thereby incurs no responsibility nor any obligation whatsoever, and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

Please do not request copies of this report from the Armstrong Aerospace Medical Research Laboratory. Additional copies may be purchased from:

National Technical Information Service  
5285 Port Royal Road  
Springfield, Virginia 22161

Federal Government agencies and their contractors registered with the Defense Technical Information Center should direct requests for copies of this report to:

Defense Technical Information Center  
Cameron Station  
Alexandria, Virginia 22314

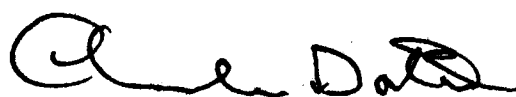
## TECHNICAL REVIEW AND APPROVAL

AAMRL-TR-88-041

This report has been reviewed by the Office of Public Affairs (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

**FOR THE COMMANDER**

  
**CHARLES BATES, JR.**  
Director, Human Engineering Division  
Armstrong Aerospace Medical Research Laboratory

UNCLASSIFIED

## SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188		
1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS			
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited			
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE						
4. PERFORMING ORGANIZATION REPORT NUMBER(S)			5. MONITORING ORGANIZATION REPORT NUMBER(S) AAMRL-TR-88-041			
6a. NAME OF PERFORMING ORGANIZATION University of Illinois *		6b. OFFICE SYMBOL (if applicable)	7a. NAME OF MONITORING ORGANIZATION Harry G. Armstrong Aerospace Medical Research Laboratory			
6c. ADDRESS (City, State, and ZIP Code) Champaign IL 61820			7b. ADDRESS (City, State, and ZIP Code) Wright-Patterson AFB OH 45433-6573			
8a. NAME OF FUNDING / SPONSORING ORGANIZATION		8b. OFFICE SYMBOL (if applicable) HEG	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F33615-84-D-0505			
8c. ADDRESS (City, State, and ZIP Code)			10. SOURCE OF FUNDING NUMBERS			
			PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
			61102F	2313	V1	35
11. TITLE (Include Security Classification) The Retrieval of Information from Secondary Memory: A Review and New Findings (U)						
12. PERSONAL AUTHOR(S) Strayer, David L. and Kramer, Arthur F.						
13a. TYPE OF REPORT FINAL		13b. TIME COVERED FROM Sep 87 TO Mar 88		14. DATE OF REPORT (Year, Month, Day) 1988 March		
15. PAGE COUNT 181						
16. SUPPLEMENTARY NOTATION * Subcontract to Universal Energy Systems, 4401 Dayton-Xenia Rd., Dayton, Ohio 45432						
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)			
FIELD	GROUP	SUB-GROUP	Evoked Potentials			
05	08		Memory Search			
06	04		Reaction Time			
19. ABSTRACT (Continue on reverse if necessary and identify by block number) Memory search was compared when the memorized items were either in primary or secondary memory. Distractor tasks were used to require secondary memory storage during a memory retrieval task. The additive effects of memory load and delay support the interpretation of separate retrieval and memory search processes. In a second experiment, event related potentials were used to examine the hypothesis that the increase in reaction time from primary to secondary memory was due to the insertion of a retrieval process prior to memory comparison. P300 latency data suggest that stimulus evaluation and response related processing are both affected by delay.  <div style="text-align: right;"> <i>Keywords: brain evoked potentials</i>  <i>Brain (ERP)</i> </div>						
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED			
22a. NAME OF RESPONSIBLE INDIVIDUAL GLENN F. WILSON, Ph.D.			22b. TELEPHONE (Include Area Code) 513-255-8748		22c. OFFICE SYMBOL AAMRL/HEG	

## PREFACE

The purpose of the project summarized by this report was to investigate the processes involved in retrieving information from secondary memory. Reaction time and evoked potential methods were used as measures of performance. The strategy applied was to compare differences in these measures between situations utilizing primary and secondary memory tasks.

While reaction times represent the total time taken to evaluate and respond to a given stimulus in a task, the brain evoked potential may be able to shed light on the nature of the component processes that yield the final motor responses measured with the reaction time. This window into the "black box" could be very helpful in illuminating the underlying procedures that the human brain utilizes in processing information.

Initial phases of this project were carried out in collaboration with Dr. Delos Wickens and his input and insights are gratefully acknowledged.

<b>Accession For</b>	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	



# TABLE OF CONTENTS

	Page
Experiment 1: Primary/Secondary Memory Search . . . . .	4
Methods: Experiment 1 . . . . .	5
Results and Discussion: Experiment 1 . . . . .	7
Conclusions: Experiment 1 . . . . .	10
Experiment 2: ERP Analysis of Primary/Secondary Memory Search . . .	12
Methods: Experiment 2 . . . . .	12
Results and Discussion: Experiment 2 . . . . .	14
Conclusions: Experiment 2 . . . . .	30
Footnotes . . . . .	32
References . . . . .	33
Appendix A . . . . .	36
Appendix B . . . . .	60
Appendix C . . . . .	62
Appendix D . . . . .	65
Appendix E . . . . .	69
Appendix F . . . . .	71
Appendix G . . . . .	75
Appendix H . . . . .	80
Appendix I . . . . .	101
Appendix J . . . . .	122
Appendix K . . . . .	154
Appendix L . . . . .	172
Appendix M . . . . .	177
Appendix N . . . . .	180

Several studies have compared performance in memory search tasks when the memorized items are either in primary or secondary memory (see footnote 1). The focus of these studies is on the dynamics of the process of retrieval of information from secondary memory to primary memory. Using additive factors (Sternberg, 1969b) and subtraction (Donders, 1868/1969) logic, one can make inferences about the processes which are involved in the transfer of information and the relative duration of these processes. This can be achieved by comparing the regression equation parameters of performance in primary and secondary memory. Differences in the intercepts and slopes of primary and secondary memory reveal the characteristics of this transfer process. Typically these experiments employ a distractor task between the presentation of the memory set and the probe stimulus to prevent rehearsal of the memory set items. Without rehearsal, information in primary memory decays rapidly (e.g., Murdock, 1961; Muter, 1980; Peterson & Peterson, 1959), thus the information must be retrieved from secondary memory for subjects to perform the memory search task.

Sternberg (Sternberg, 1969a, exp. 5; Sternberg, Kroll, and Nasto, 1969) was one of the first to compare primary and secondary memory performance in the memory search task. Subjects memorized a list of 1, 3, or 5 digits. In the secondary memory condition, a list of 7 letters was sequentially presented (with a total duration of 3.5 seconds) prior to the presentation of the probe stimulus. Subjects were instructed to retain the list of letters in memory, and catch trials were included to insure that subjects complied with instructions. The retention of the list of letters was intended to prevent subjects from rehearsing the memory set and require the retrieval of the memory set from secondary memory. In the primary memory condition, no distractor task preceded the probe stimulus. The linear regression equation for primary memory was  $RT = 336 + 57(x)$ , while for secondary memory  $RT = 467 + 105(x)$ , where  $x$  refers to memory set size. Both the intercept and the slope were greater for the secondary memory condition. The increase in the intercept (131 msec) was taken to reflect the time to locate the memory set in secondary memory. The increase in the slope (48 msec) was taken to suggest a serial transfer of the entire memory set into primary memory. One caveat in interpreting the differences between primary and secondary memory in these studies concerns the duration of the distractor task. Evidence from the Brown-Peterson paradigm suggests that the memory set information may not have completely decayed from primary memory within the 3.5 second interval (see Flexser, 1978 for a similar argument). If this were the case, the increase in the slope from primary memory to secondary memory may be the result of a mixture of primary and secondary memory performance, and thus the slope differences are ambiguous. Furthermore, if the memory set information had not fully decayed from primary memory, the difference between primary and secondary memory would underestimate the true retrieval time.

Wickens and colleagues (Coyne, Allen, and Wickens, 1986; Wickens, Moody, and Dow, 1981; Wickens, Moody, and Vidulich, 1985) have provided the most extensive studies comparing primary and secondary memory. Wickens et al. (1981, exp. 1) presented memory sets of 2 or 4 words for 3 seconds. Following the memory set, subjects were presented with a random three digit number and instructed to count backwards by 3s for a duration of 12 seconds. A 2 second interval in which the random number was removed from the display signaled the end of the distractor task and the upcoming probe stimulus. No

measure of subjects' performance was obtained for the distractor task. The probe was presented for 2 seconds, and the interval in which subjects could respond was 3 seconds from probe onset. A manipulation investigating the effects of proactive interference on primary and secondary memory was accomplished by selecting the words in the memory set from the same taxonomic category. Three trials using stimuli from the same taxonomic category were presented (without repetition of words) before a new category was chosen. The first memory set was deemed low in interference, the third high in interference. Experimental factors were mixed within the session. Wickens et al. (1981, exp. 2) was identical to Wickens et al. (1981, exp. 1), with the exception that the distractor task was eliminated. This constituted the primary memory condition. Note that the primary - secondary memory manipulation was a between subjects factor. Thus comparisons between experiments 1 and 2 provide a measure of the time to transfer the memory set from secondary memory to primary memory.

The slopes and intercepts obtained in the experiment were:

	Primary	Secondary
target	$RT = 500 + 33(x)$	$RT = 595 + 39(x)$
non-target	$RT = 501 + 36.25(x)$	$RT = 630 + 37.5(x)$

It is evident that memory load did not interact with the manipulation of primary and secondary memory, which is in contrast to Sternberg's (1969a, exp. 5) results. One potential reason for the discrepancy is the nature and duration of the distractor task. The Wickens study prevents subjects from rehearsing the memory set for a longer duration than Sternberg's studies, and probably reflects a more accurate representation of secondary memory performance. It is also apparent that there is a substantial difference in the intercepts of primary and secondary memory (112 msec). This estimate is more closely in agreement with the Sternberg studies.

The data from experiments 1 and 2 were fitted to the following regression equation:

$$RT = [ a + b(x) + t ] + [ ( r1 + r2 ) + q ]$$

The first bracketed term in the equation describes performance in primary memory, where " a " refers to the intercept, " b " refers to the slope, " x " refers to memory set size, and " t " refers to differences between target and non-target stimuli. The second bracketed term describes performance in secondary memory. The term " r1 " refers to the increase in the intercept under low proactive interference conditions, " r2 " refers to the additional increase under high proactive interference conditions, and " q " refers to differential effects for target and non-target stimuli. The values for the terms in the equation are:

$$RT = [ 489 + 37(x) + 14 ] + [ ( 95 + 38 ) + 17 ]$$

This model accounted for 99% of the variance across conditions. It is important to note that these results imply that the retrieval process is independent (but see Sternberg, 1969b) of the memory search process. According to this interpretation, there is a constant time for retrieving the memory set items, and they are retrieved in a chunk (i.e., in parallel). Furthermore, the effects of memory load produced equivalent results, suggesting that the memory search process is the same for items in primary memory as for items which have to be transferred from secondary memory to primary memory.

Wickens (Wickens et al., 1985, exp. 1; Wickens, personal communication, March 12, 1987) has also demonstrated the independence of memory search rate and memory retrieval time using both out of category non-target probes and repeated stimuli. Both effects reduced the slope, but the primary - secondary memory differences were maintained. Flexser (1978) has also reported similar intercept differences between primary and secondary memory using memory sets of 16 and 32 (which produced small effects on search time). These results offer additional support for the assertion that the retrieval process and the memory search process are independent. Wickens concluded that secondary memory conditions differ from primary memory conditions in the addition of a retrieval process inserted prior to the memory search operation.

In sum, the studies which have compared primary and secondary memory using memory search paradigms have universally found that there is an increase in the intercept between primary and secondary memory. The magnitude of this effect seems to vary with the type of material — words produce a larger effect than letters or digits (e.g., Wickens et al., 1985, exp. 2). Furthermore, there is some evidence that the magnitude of the difference between primary and secondary memory is dependent on the strength of the trace in primary memory. The longer the interval between the memory set and the probe stimulus, the greater the difference between primary and secondary memory. This difference in the intercept between primary and secondary memory has been interpreted as the time to retrieve the information from secondary memory.

The effect of memory load on primary and secondary memory has produced mixed results. The majority of studies report parallel curves for primary and secondary memory (Wickens et al., 1981; Wickens et al., 1985; Wickens, personal communication, 1987; Flexser, 1978). Forrin and Morin (1969) reported shallower slopes for the secondary memory condition, but subjects received substantially more practice (i.e., consistent practice) in the secondary memory condition than in the primary memory condition. A few studies have reported steeper slopes for secondary memory conditions (e.g., Sternberg, 1969; Sternberg, Kroll, and Nasto, 1969; Peters, 1974). However, in these studies the interval between the memory set and the probe stimulus was such that it is questionable whether the memory set information had decayed from primary memory. If one assumes that rehearsal strengthens the trace in primary memory, as set size increases the average trace strength of each memory set item should decrease. This would result in differential decay rates for different memory set sizes. Smaller memory set sizes should take longer to decay, since the rehearsal cycle time varies with memory load (e.g., Baddeley and Ecob, 1973; Cavanaugh, 1972; Clifton and Birenbaum, 1970; Corballis, Kirby, and Miller, 1972; Monsell, 1978). This would



produce a greater proportion of primary memory trials following a short distractor task for smaller set sizes than for larger memory set sizes. The expected outcome would result in a steeper slope for this mixture condition than for a pure primary memory or a pure secondary memory condition.

An important consideration when evaluating the differences between the studies concerns the nature of the distractor task. The distractor tasks varied in difficulty and duration, and none of the experiments objectively evaluated subjects' performance in the distractor task. The most effective distractor task employed a backwards count task for a duration of at least 12 seconds. All of these studies reported parallel memory load functions for primary and secondary memory. However, since performance in the distractor task was not monitored, it is possible that subjects may have rehearsed the memory set even in these conditions. Clearly a distractor task in which performance can be monitored over the course of the distractor interval is warranted.

A final issue concerns the interval between when the distractor task was terminated and when the probe stimulus was presented. This duration varied from 500 msec to 2 seconds in the studies reviewed above. It is unclear what subjects were doing in this interval, and why they did not transfer the memory set information from secondary to primary memory. Given the estimates of retrieval time, subjects should have had ample time to complete this operation prior to the presentation of the probe stimulus. Upon post-experimental questioning, subjects reported that they did not retrieve the memory set information prior to the presentation of the probe stimulus (Wickens et al., 1985). If subjects had, in fact, retrieved any information from secondary memory, this should tend to obscure any true differences between primary and secondary memory.

#### Experiment 1: Primary/Secondary Memory Search

Several methodological and interpretative issues remain which need to be addressed before this paradigm can be used to assess the dynamics of memory retrieval. The purpose of experiment 1 was to address these issues and provide additional information concerning the process of retrieval of information from secondary memory to primary memory.

One issue to be addressed is the proposition that the retrieval process is independent from the memory search process. There was a trend in the literature for studies using short distractor intervals (e.g., 4 seconds) to find larger slopes for secondary memory conditions, while studies which used longer distractor intervals (e.g., 12 seconds) found equivalent slopes for primary and secondary memory. The present experiment will evaluate the contribution of the duration of the distractor interval on the additivity of the retrieval and memory comparison processes by manipulating the delay between the memory set and the presentation of the probe stimulus. Three delay conditions will be included: 0, 4, or 15 sec. The 0 delay condition represents the situation in which no distractor task is presented between the memory set and the probe (i.e., a pure primary memory condition). In the 4 second delay condition, subjects will perform a distractor task for 4 seconds prior to the presentation of the probe stimulus. The duration of the distractor task in this condition is quite close to the parameters of Sternberg's experiment in which an increase in the set size slope was

observed compared to primary memory. In the 15 second delay condition, subjects will perform a distractor task for 15 seconds prior to the presentation of the probe. This condition is roughly equivalent to that employed in the Wickens studies, and is intended to represent a condition in which the memory set information is not in primary memory (i.e., a pure secondary memory condition). Following additive factors logic, if the memory retrieval process is independent of the memory search process, then the effects of delay should be additive with the effects of memory load.

A second issue to be addressed by this research is related to the finding in the Wickens study that subjects apparently did not retrieve the memory set until the probe was presented, even though they were cued to stop the distractor task and prepare for the probe trial with a 2 second warning interval. It is unclear why subjects did not retrieve the memory set during this interval. If subjects retrieved any information prior to the presentation of the probe stimulus, the estimates of retrieval time represent an underestimate of the true retrieval time. The present study will present a retrieval cue which signals the termination of the distractor task (if presented). The probe stimulus will be presented at three stimulus onset asynchronies (SOAs) following retrieval cue onset: 200, 500, and 1000 msec. If subjects are retrieving any of the memory set information prior to the presentation of the probe, then the retrieval time estimate (the intercept difference between primary and secondary memory) should diminish as SOA increases. The three SOA conditions will be factorially combined with the three delay conditions. Two memory set sizes will be presented: 2 and 4 and target and non-target trials will be equiprobable. All conditions will be mixed to reduce changes in bias between conditions. It is predicted that reaction time will increase as a function of delay and that this increase reflects the time to retrieve the memory set information from secondary memory. It is also predicted that if the transfer of the memory set information is parallel, as suggested by Wickens, then the slopes for the 0 and 15 sec delay conditions should be equivalent. If performance in the 4 sec delay condition reflects a mixture of primary and secondary memory conditions and this proportion varies with memory load, then there should be an increase in the slope for this condition relative to the 0 and 15 sec delay conditions. These effects should be modulated by SOA. The largest differences between delay conditions should be observed with the shortest SOA condition. If subjects retrieve any information prior to the presentation of the probe stimulus, the difference between delay conditions should decrease as SOA increases.

#### Methods: Experiment 1

##### Subjects

Ten subjects (3 female) age 18 to 25 participated in experiment 1. Each subject participated in 6 one hour sessions. Subjects were paid for their participation.

##### Stimuli

The stimuli presented in the Sternberg portion of the experiment were 4 letter, monosyllabic nouns with a word frequency of 70 to 148 (Kucera, Henry, Francis, and Nelson, 1967). Words were selected so as not to rhyme

or have orthographic similarity. A further constraint used in stimulus selection was to examine all possible word pairs and eliminate any words which resulted in strong natural associations. Thus the words formed a heterogeneous class of stimuli. Appendix E presents the word-list employed in the experiment. The approximate visual angle of the words subtended 1.2 degrees horizontally and 0.5 degrees vertically.

The stimuli used in the recognition running memory portion of the experiment were the digits 1 to 9. Digits were selected randomly. The approximate visual angle of the running memory stimuli subtended 0.3 degrees horizontally and 0.5 degrees vertically.

### Apparatus

The experiment was performed on an IBM XT, with a quadEGA card which permitted cursor control and synchronization. The stimuli were displayed on an IBM monochrome display. Subjects indicated their responses by pressing the "Z" key with the left index finger and the "/" key with the right index finger on the keyboard of the IBM computer.

### Procedure

Each trial was comprised of the following events. A memory set was presented for a duration of 3000 msec. This was followed by a 1500 msec interval in which the display was blanked. Three delay intervals followed the memory set: 0, 4, and 15 seconds. During the delay interval, subjects performed a recognition running memory task which prevented rehearsal of the memory set. Following the delay interval, an asterisk was presented for 200 msec which served as a cue for subjects to retrieve the memory set information. Three stimulus onset asynchronies (SOAs) were included between onset of the retrieval cue and the onset of the Sternberg probe stimulus: 200, 500, and 1000 msec. The Sternberg probe was presented for 200 msec and legal reaction times were permitted within a 3000 msec interval following probe onset.

Sternberg memory set sizes of 2 and 4 words were used in the experiment. Target and non-target trials were presented equiprobably. Targets were defined as items from the memory set. Non-targets were items not included in the memory set. Subjects pressed one key for target trials and another key for non-target trials. Key assignments were counterbalanced across subjects. Instructions emphasized both speed and accuracy.

The recognition running memory task was performed in the interval between the presentation of the memory set and the retrieval cue. The task consisted of a series of digits presented successively for 200 msec, with an interstimulus interval of 1000 msec. The subject's task was to press one button if the digit presented on trial N matched the digit presented on trial N-2 and another button if it did not match the item. It is important to note that each stimulus served as a probe in the recognition task, and subsequently served as a template against which digits presented two trials later must be compared. For subjects to successfully perform the task, it is necessary for them to maintain the last two digits in memory. Digits were chosen randomly, with the constraint that mismatched stimuli occurred twice as often as matched stimuli. This constraint was introduced to keep the

difficulty of the running memory task constant across trials. Subjects were given 1000 msec to indicate their response. Instructions emphasized both speed and accuracy. It should be re-emphasized that subjects found the running memory task extremely demanding, requiring all their effort to maintain performance in the task. While subjects performed the running memory task, they were instructed not to rehearse or otherwise think about the memory set items. Performance in the running memory task provided a good index of subject's compliance to these instructions (see footnote 2).

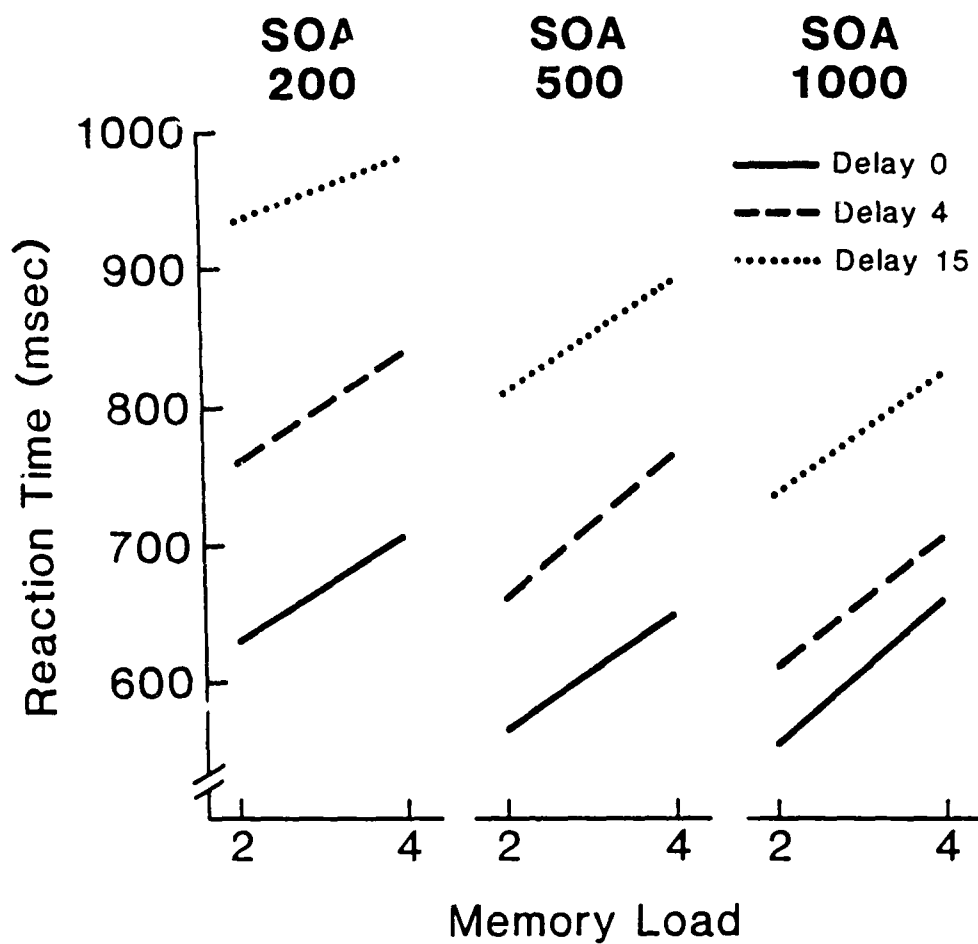
### Design

Memory set sizes of 2 and 4 were presented, with target and non-target trials equiprobable. Three delay conditions were included: 0, 4, and 15 seconds. During the delay, subjects performed the recognition running memory task, which served to prevent subjects from rehearsing the memory set. In addition, three SOAs between retrieval cue onset and probe stimulus onset were employed: 200, 500, and 1000 msec. Conditions were factorially combined to form a 2 (memory load) X 2 (target vs non-target) X 3 (delay) X 3 (SOA) design. Conditions were randomly permuted across sessions and subjects. Subjects participated in all experimental conditions. A total of 36 observations per cell were obtained for each subject. The first session of the experiment served as practice and was not included in the analyses.

### Results and Discussion: Experiment 1

Figure 1 presents the mean reaction times obtained in the Sternberg task. Only trials in which subjects were correct are included. An additional constraint employed in selecting trials for analysis omitted trials in which performance in the running memory task (if presented) was below chance accuracy. Data are collapsed across target and non-target trials for the three different SOA conditions. Line segments connect memory set size means within a condition. Several effects are noteworthy. First, reaction time increased as a function of memory load in all conditions,  $F(1,9)=36.56$ ,  $p<.001$ ,  $MSE=17194$ . Memory load did not interact with any other variables (all  $ps > .10$ ), suggesting that the memory search process did not differ between tasks. This conclusion is borne out by the linear regression slopes fitted to each of the conditions in the experiment. The average slope in the experiment was 42.0 msec per item. A comparison of the 0 and 15 second delay conditions replicates the effects reported by Wickens. The hypothesis that the difference in slopes between primary and secondary memory reported by Sternberg was due to the difference in the duration of the distractor task was rejected. This follows given the equivalence of the slopes across all delay conditions. Furthermore, these data are consistent with a model in which the retrieval process is independent of the memory search process, since memory load did not interact with delay.

A second noteworthy aspect of the data is the effect of SOA on reaction time. There was a general trend for reaction time to decrease as SOA increased,  $F(2,18)=62.93$ ,  $p<.001$ ,  $MSE=8008$ . We interpret this change as a non-specific warning effect. The greater the time between the onset of the retrieval cue and the onset of the probe, the more subjects prepared for the upcoming probe. This effect was non-monotonic, with the greatest effect obtained between the 200 and 500 msec SOA conditions (67 msec), and a lesser effect between the 500 and 1000 msec SOA conditions (42 msec).



Mean reaction time for each condition in experiment 1. The data are plotted as a function of memory set size for each delay and SOA condition. The left column in the figure represents the 200 msec SOA condition, the center column represents the 500 msec SOA condition, and the right column represents the 1000 msec SOA condition.

FIGURE 1

The manipulation of delay produced large differences in performance. As delay increased, reaction time increased,  $F(2,18)=18.26$ ,  $p<.001$ ,  $MSe=91,876$ . This effect produced changes in the linear regression intercept, but not the slope. Following Sternberg's additive factors logic, the additivity suggests that the delay and memory loads affect different processes. The former presumably affects a retrieval process, while the latter affects a memory search process.

The effect of delay was modulated by SOA,  $F(4,36)=9.81$ ,  $p<.001$ ,  $MSe=3594$ . As SOA increased, the effect of delay decreased. However, this reduction in reaction time occurred largely between the 0 and 4 second delay conditions, while the difference between the 4 and 15 second delay conditions remained constant across SOA conditions. The difference between the 0 and 15 second delay conditions was approximately 285 msec in the 200 msec SOA condition. This decreased to approximately 175 msec in the 1000 msec SOA condition.

The difference in delay conditions has been used to infer the duration of the retrieval process. Since the magnitude of the effect of delay decreases as SOA increases, it suggests that part of the memory set information is retrieved during the SOA interval. It is unclear why subjects retrieved only part of the information during the SOA interval. Given the estimate of retrieval time (275 msec) and the time to process the retrieval cue (upperbound estimate of 300 msec from simple RT tasks), subjects should have been able to retrieve the memory set information within 600 msec. Thus subjects are not fully capitalizing on the SOA interval.

One reason that reaction time differed as a function of delay in the 1000 msec SOA condition might be that this condition represents a mixture of primary and secondary memory trials. On some trials, the memory set information may be retrieved, while in others the retrieval may not have been completed. If the proportion of primary memory trials to secondary memory trials increased as SOA increased, the effect of delay should diminish as SOA increases, which is precisely what is observed in the data. If this hypothesis is correct, then the 1000 msec SOA reaction time distribution should represent a mixture of pure primary and pure secondary memory trials.

To address the mixture hypothesis, the reaction time distributions for each subject and each condition were vincentized (Vincent, 1912; Ratcliff, 1979) to form a composite cumulative distribution function (CDF). The vincentizing procedure is described in detail by Ratcliff (1979). Briefly, the reaction times for each subject and condition are sorted into ascending order and the quantiles are calculated. The quantiles are then averaged across subjects to obtain the group quantiles. From the group quantiles a group reaction time distribution is generated which retains the shape of the individual subject distributions. This process is equivalent to a simple linear interpolation, and the resulting distribution represents the distribution of the average subject. The CDFs have been divided into 20 intervals, each containing 5% of the distribution. The mean of each interval is cross-plotted against the interval position.

The CDFs for the 0 and 15 second delays are plotted at each SOA for target memory load 4 trials in figure 2. All the CDFs have a general

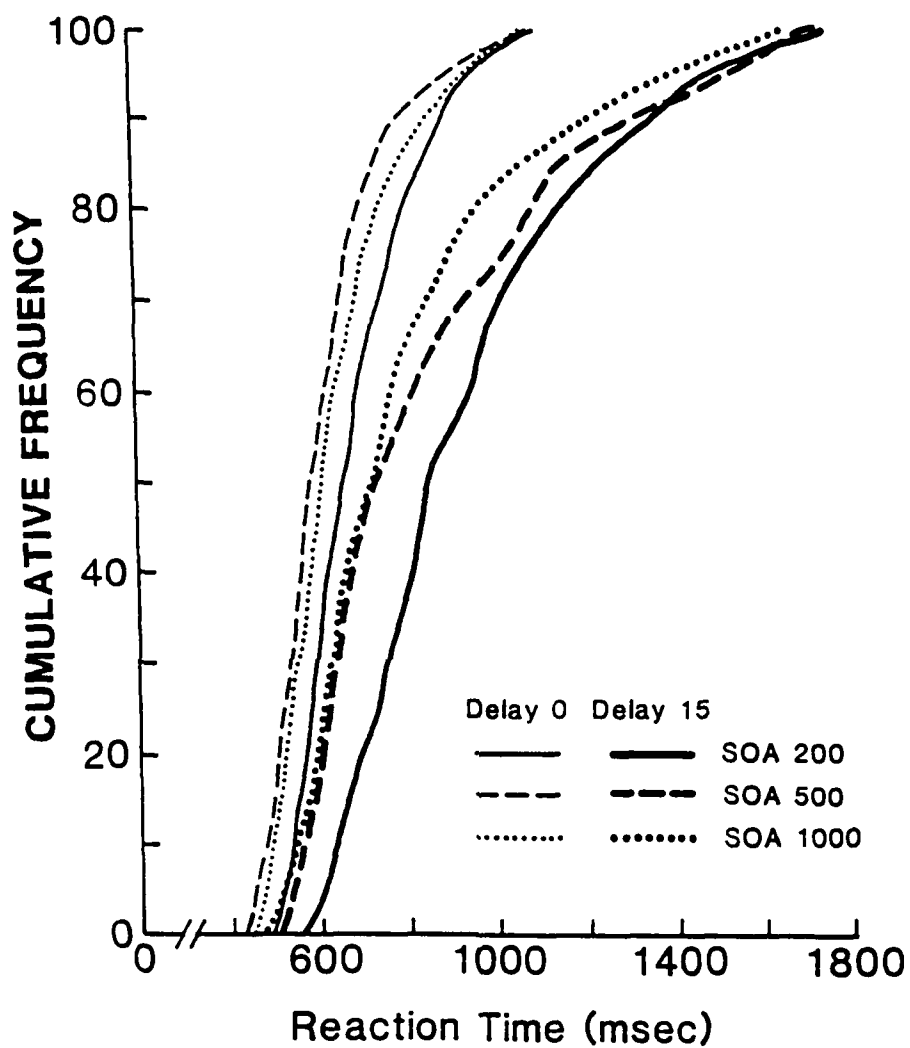
scallop shape which reflects a positive skew in the RT distributions. Of central interest is the shape of the distributions for the different delays as a function of SOA. For the 0 delay condition, the CDFs are tightly clustered and parallel over the entire latency interval. Each distribution is shifted with SOA. The CDFs for the 15 second delay conditions are also plotted in the figure. For the 200 msec SOA condition the entire CDF has shifted out in time relative to the 0 delay CDFs. In contrast, the 500 and 1000 msec SOA conditions produced CDFs which initially clustered with the 0 delay conditions. At about the median of each CDF, the functions deviate from the 0 delay conditions and approach the 15 second delay, 200 msec SOA condition. This reflects a mixture of distributions (Ratcliff, 1979) and supports the hypothesis that the 1000 and 500 msec SOA conditions reflect a mixture of primary and secondary memory trials. Furthermore, if subjects are retrieving memory set information in the SOA interval, one would predict that the proportion of primary memory trials would be larger in the 1000 msec SOA condition. This in fact appears to be the case. The CDF for the 500 msec SOA condition deviates from the 0 delay conditions more rapidly than does the 1000 msec SOA condition. Thus, these data suggest that subjects are retrieving information about the memory set during the SOA interval and that performance at longer SOAs reflects a mixture of primary and secondary memory trials. The proportion of primary to secondary memory trials is modulated by the SOA interval.

To estimate the proportion of primary and secondary memory trials in the 1000 msec SOA, 15 second delay condition, hypothetical distributions were generated by randomly selecting trials from the 0 and 15 second delay, 200 msec SOA conditions. Figure 3 presents the CDFs for selected proportions of primary and secondary memory trials. The CDF obtained in the 1000 msec SOA, 15 second delay condition (adjusted for SOA) is also plotted in figure 3. This condition falls between 50/50 and 75/25 secondary/primary memory proportions and provides further support for the mixture hypothesis.

One effect which did not attain significance was the effect of response type (target vs non-target),  $F(1,9)=3.06$ ,  $p>.10$ ,  $MSe=88852$ . This is consistent with previous research in which the memory set is changed after every trial (Sternberg, 1975). Further, the probability of a mismatched trial was twice the probability of a matched trial in the running memory task and this may have offset any response bias for target trials in the Sternberg task.

#### Conclusions: Experiment 1

In sum, the additive effects of memory load and delay provide support for the interpretation of separate retrieval and memory search processes. The data further suggest that the memory set information is retrieved as a unit. These results are in direct contrast to Sternberg's findings. The data from the 4 second delay condition suggest that the duration of the distractor task was not responsible for these differences. The manipulation of SOA provided evidence that subjects retrieve some of the information during the SOA interval; however, this retrieval is not complete prior to the presentation of the probe. The 1000 msec SOA condition should have provided ample time to retrieve the information. Evidence from the distributions of reaction time for the different conditions suggest that performance in the 1000 msec SOA condition may represent a mixture of primary and secondary memory conditions.



#### Target Load 4

The vincentized cumulative distribution functions for the 0 and 15 second delay conditions at each SOA condition. The target memory load 4 condition is presented. The reaction time distribution was divided into 5% quantiles and the mean latency of each quantile was calculated. The means are plotted as a function of quantile.

FIGURE 2



## Experiment 2: ERP Analysis of Primary/Secondary Memory Search

Experiment 2 examined the Event-Related Brain Potentials (ERPs) elicited in the primary/secondary memory search task. This analysis was conducted to determine if the increases in reaction time as a function of delay in experiment 1 were the result of increases in stimulus evaluation processing, response-related processing, or both. Wickens et al. (1981, 1985) hypothesized that the increase in reaction time from primary to secondary memory was due to the inclusion of a retrieval process inserted prior to the memory comparison process in secondary memory conditions. If this hypothesis is correct, then stimulus evaluation processes should be delayed as subjects must retrieve the memory set information from secondary memory. However, performance in secondary memory may be slower than primary memory due to shifts in response bias. Error rates were greater in secondary memory, and subjects may have adopted a more conservative response bias under these conditions. Since the P300 component of the ERP is sensitive to stimulus evaluation processes, but relatively insensitive to response-related processes (Magliero, Bashore, Coles, and Donchin, 1983; McCarthy and Donchin, 1981), the P300 can be used to dissociate these two factors.

The 500 msec SOA condition of experiment 1 was selected to record ERPs from experiment 2. The experimental parameters were identical to experiment 1. If P300 latency increases as a function of delay, this will provide support for the hypothesis that a retrieval process is inserted prior to the memory comparison process. If P300 latency does not increase as a function of delay, this will provide support for the hypothesis that primary and secondary memory differences are the result of response criterion shifts.

### Methods: Experiment 2

#### Subjects

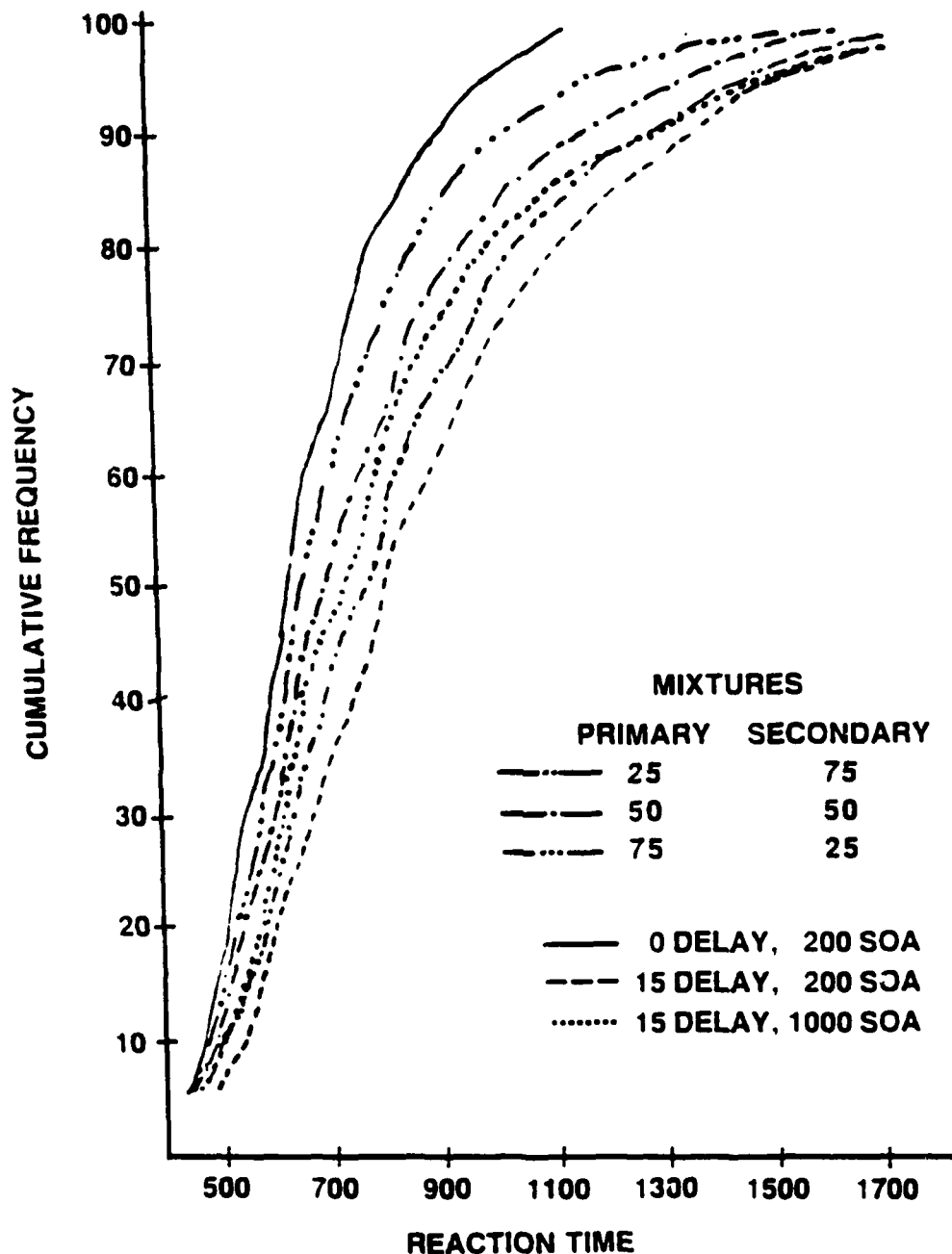
Ten young subjects (age range 18 to 30) participated in the experiment (see footnote 3). Subjects were right-handed with normal or corrected-to-normal vision. Subjects were paid for their participation in the experiment.

#### Stimuli and Apparatus

The stimuli were identical to experiment 1. The stimuli were presented on a Matrox display positioned approximately 70 cm from the subjects. The stimuli were presented within a rectangle in the center of the display. The rectangle subtended a visual angle of 1.0 degrees vertically and 1.5 degrees horizontally. Subjects indicated their response by depressing one of two buttons on a response box held with both hands. Response button assignments were counterbalanced across subjects.

#### Procedure

The experimental procedure was identical to experiment 1, with the exception that only the 500 msec SOA condition was included.



An examination of different mixtures of primary and secondary memory. The data are presented as vincentized CDFs. The solid line represents the 0 second delay, 200 msec SOA condition, the dashed line represents the 15 second delay, 200 msec SOA condition. The dotted line represents the data obtained in the 15 second delay, 1000 SOA condition (adjusted for the main effect of SOA). Three mixture conditions are presented which represent 25/75, 50/50, and 75/25 mixtures of primary and secondary memory. It is apparent that the data obtained in the 15 second delay, 1000 msec SOA condition falls between the 50/50 and 25/75 mixture conditions.

FIGURE 3

## Experimental Design

The experiment included three delay conditions (0, 4, and 15 sec), and two set sizes (2 and 4) in a Sternberg memory search task. Target and non-target trials were equiprobable. Thus, the study was a 3 (delay) X 2 (memory load) X 2 (response type) factorial design. The order of within subject conditions was counterbalanced across subjects and sessions.

## ERP Recording

The electroencephalogram (EEG) was recorded from three midline sites (Fz, Cz, and Pz according to the International 10-20 system; Jasper, 1958) and referred to linked mastoids. The ground electrode was positioned on the left side of the forehead. Electrooculogram (EOG) electrodes were placed above and below the right eye. Electrode impedances did not exceed 5 KOhms. Beckman 10 mm diameter Ag/AgCl biopotential electrodes were used at all electrode sites. Scalp electrodes were affixed with Grass EEG paste. Reference and ground electrodes were adhered with stomaseal adhesive collars.

The EEG and EOG signals were amplified by Grass amplifiers (model 7P122), and filtered on-line using a high frequency cut-off point at 35 Hz and a time constant equal to 8 sec for the high pass filter. Both EEG and EOG were sampled for 2100 msec, beginning 100 msec prior to retrieval cue onset. The data were digitized every 10 msec.

## Stimulus Generation and Data Collection

Stimulus presentation and data acquisition were governed by a PDP 11/73 computer interfaced with a matrox display. Single trial EEG and EOG were monitored on line. Digitized single trial data were stored on magnetic tape for subsequent analyses. EOG artifacts were corrected off-line (Gratton, Coles, and Donchin, 1983).

## Results and Discussion: Experiment 2

The reaction time results replicated the effects obtained in experiment 1 and will therefore only be discussed briefly. Reaction time increased as a function of memory load ( $F(1,7)=8.5$ ,  $p<.02$ ,  $MSe=4998$ ). The memory set size slope was 21.0. msec/word. Reaction time also increased as a function of delay ( $F(2,14)=9.1$ ,  $p<.01$ ,  $MSe=9.1$ ). The increase in reaction time from the 0 to the 15 msec delay condition was 126 msec. In contrast to experiment 1, targets were responded to faster than non-targets ( $F(1,7)=138.6$ ,  $p<.001$ ,  $MSe=1403$ ).

Figure 4 a-f presents the grand average ERPs collected in the experiment. Each panel presents one of the twelve experimental conditions. A description of the experimental condition is presented above each panel. Within each panel the ERPs for each electrode site are over-plotted. Fz is represented by the solid line, Cz by the dashed line, and Pz by the dotted line. The abscissa represents latency, in milliseconds, from presentation of the retrieval cue (S1). The recording epoch begins 100 msec pre S1. The average of this 100 msec interval (within each electrode site) is used to remove baseline shifts. The dashed vertical line indicates the onset of the

retrieval cue (S1). The dotted vertical line represents the onset of the Sternberg probe stimulus (S2). The ordinate represents amplitude in microvolts. Positive changes are reflected as a downward deflection. The number in the upper left-hand corner of each panel indicates the number of single trials in the average.

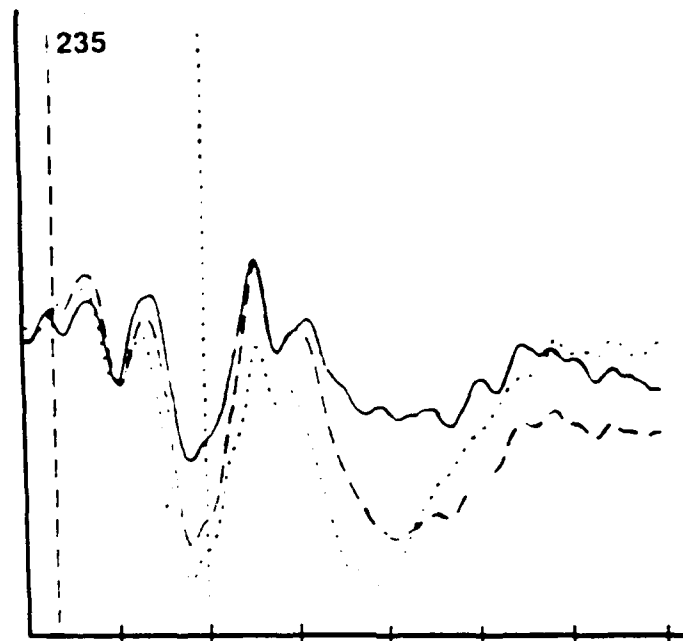
Perusal of the ERPs reveals several distinct ERP components. Following S1 onset, an early negative deflection appears in several of the tracings. The latency of this component is between 100 and 110 msec and the component is frontal-central maximal. We identify this component as the N100. The second ERP component clearly identifiable following S1 is P200, a positive deflection occurring approximately 200 msec after S1 onset. The third ERP component is N200, a negative deflection in the ERP following P200 with frontal-central maximal distribution and a latency from 250 to 350 msec. The last ERP component elicited by the processing of the information conveyed by S1 is the P300. The P300 component has a parietal maximal distribution with a minimum latency of 300 msec. P300 latency to S1 occurs between 450 to 500 msec following S1 onset.

Examination of the ERP components elicited by the processing of the Sternberg probe (S2) is complicated somewhat by baseline shifts due to S1 processing. To compensate for this, the ERPs were reaveraged and a new baseline interval from onset of S2 to 100 msec post S2 onset was used to remove baseline differences. Figures 5 a-f present the reaveraged ERPs. It should be noted that the different baselines do not result in changes in the ERP component structure, nor in estimates of the latency of these components. However, without a proper baseline, any base to peak amplitude measures will be biased.

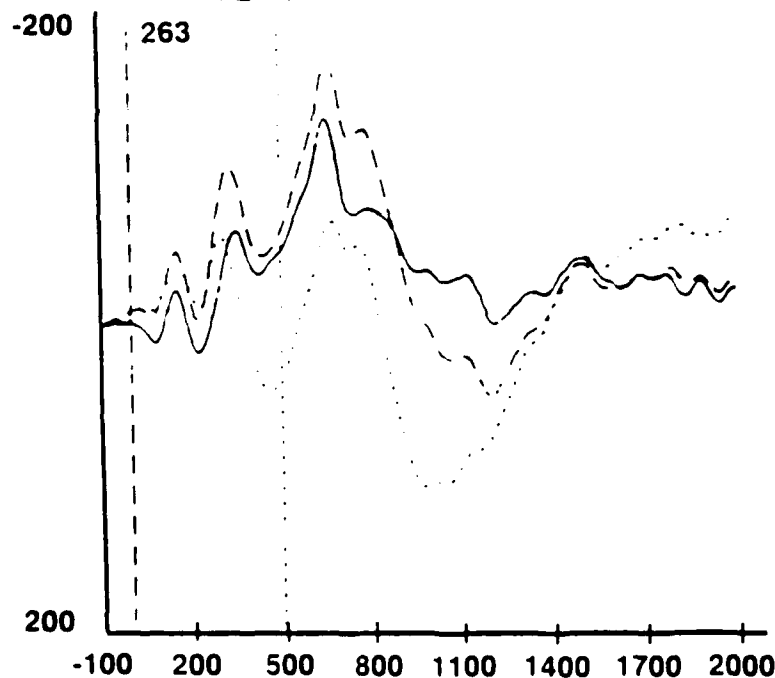
Examination of the ERP components following S2 reveals a component structure similar to S1. The components are N100 (a negativity with a latency from 100 to 110 msec post S2), P200 (a positivity with a latency from 200 to 250 msec which is somewhat more difficult to identify, compared with the P200s elicited by S1), N200 (a negativity with a latency from 200 to 350 msec), and P300 (a positivity with a latency from 300 to 700 msec). It should be noted that the latency values are estimates of the average latency within a condition. The range of the P300 on single trials is from 300 to 1200 msec.

Because the focus of experiment 2 was to dissociate stimulus evaluation processing from response-related processing, our analyses will focus on the P300 component of the ERP. We have therefore overplotted the Pz ERPs for the different delay conditions in figures 6 and 7. The baseline used for figure 6 a-b is from 100 msec pre S1 to S1. The baseline used for figure 7 a-b is from S2 to 100 msec post S2. A description of the experimental condition appears above each panel. The solid line represents the 0 delay condition, the dashed line represents the 4 second delay condition, and the dotted line represents the 15 second delay condition. To examine latency and amplitude differences in the P300, an estimate was derived for each single trial. The P300 was identified by computing the correlation between the positive segment of a .5 Hz sine wave and the Pz electrode (described as Pz cross correlation in appendix K). The correlations were computed at 10 msec lags within an interval from 300 to 600 msec post S1 and 300 to 1200 msec post S2 for the S1 and S2 P300s, respectively. The point at which the

# MEMSIZ 2 DELAY 4 TARGET



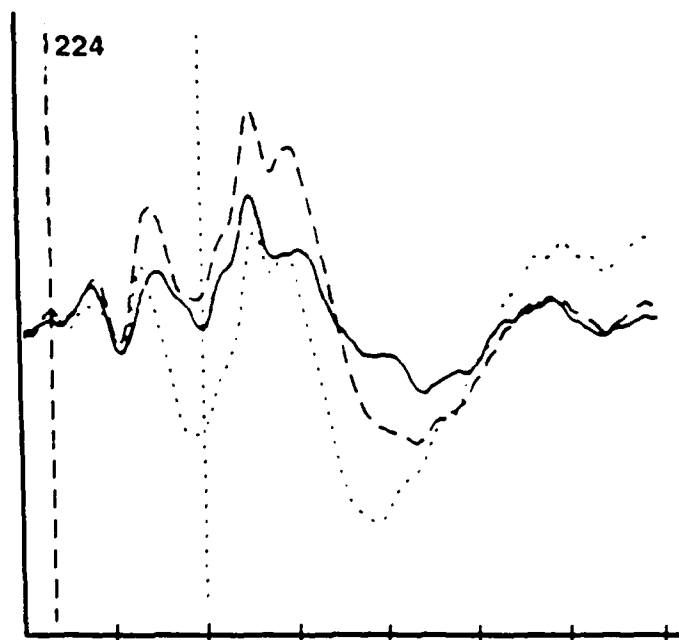
# MEMSIZ 2 DELAY 0 TARGET



Grand average scalp distribution ERPs from experiment 2. Ten subjects are included in the average. The baseline subtracted from each waveform is from 100 msec pre-S1 to S1. The number of trials included in each average is displayed in the upper left-hand portion of each plot. The solid line represents the Fz electrode, dashed, Cz, and dotted Pz. The experimental condition is presented above each panel. Figures a-f contain the 12 experimental conditions.

FIGURE 4 A

MEMSIZ 4 DELAY 0 TARGET



MEMSIZ 2 DELAY 15 TARGET

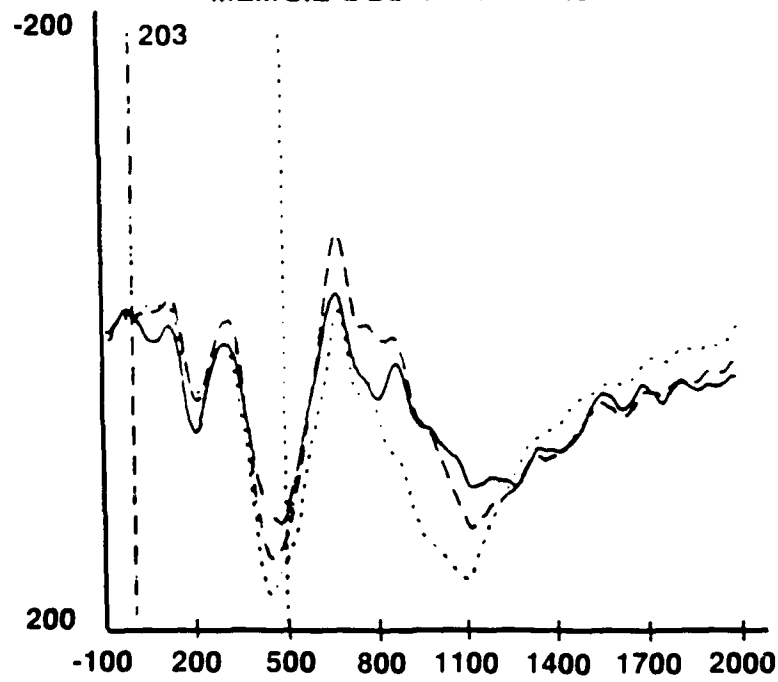
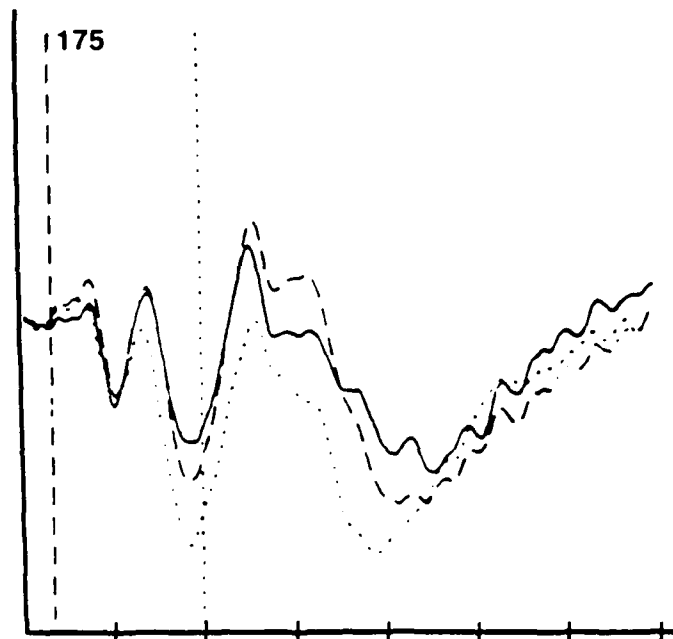


FIGURE 4 B

MEMSIZ 4 DELAY 15 TARGET



MEMSIZ 4 DELAY 4 TARGET

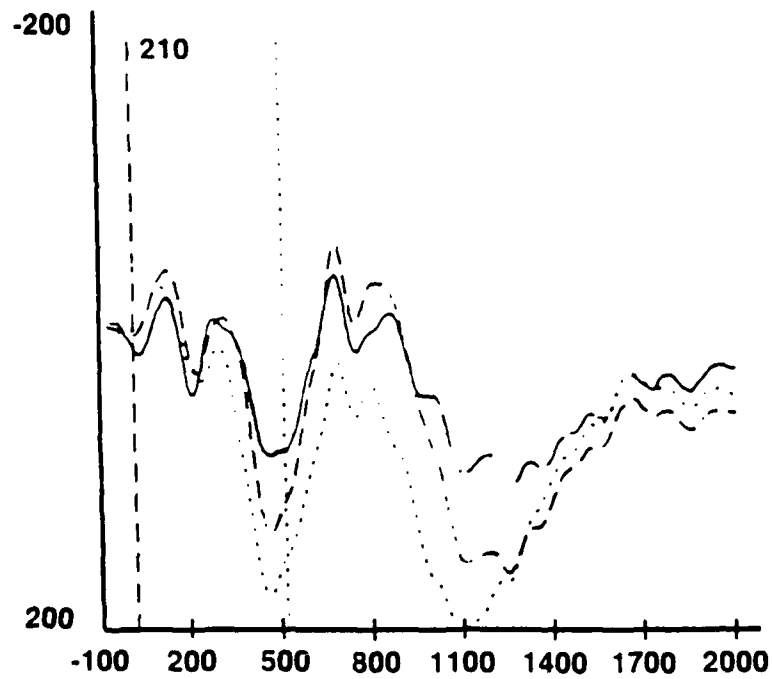
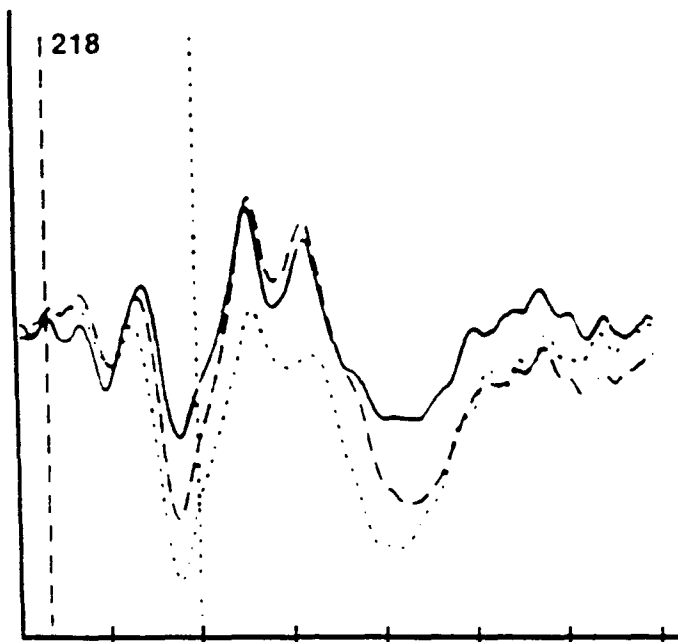


FIGURE 4 C

MEMSIZ 2 DELAY 4 NONTARGET



MEMSIZ 2 DELAY 0 NONTARGET

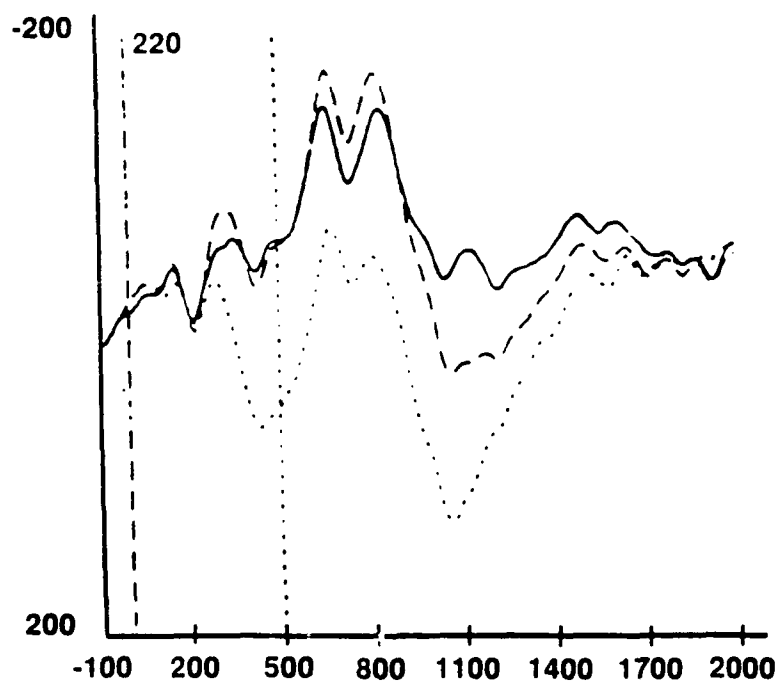
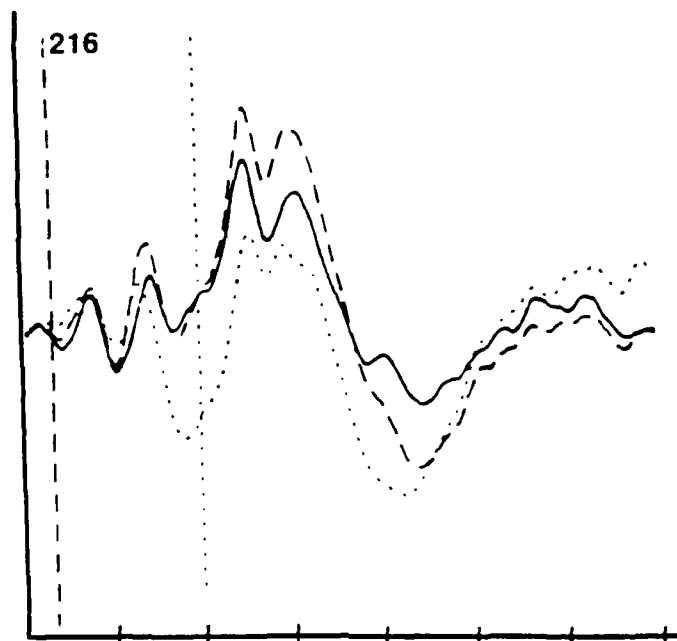


FIGURE 4 D



MEMSIZ 4 DELAY 0 NONTARGET



MEMSIZ 2 DELAY 15 NONTARGET

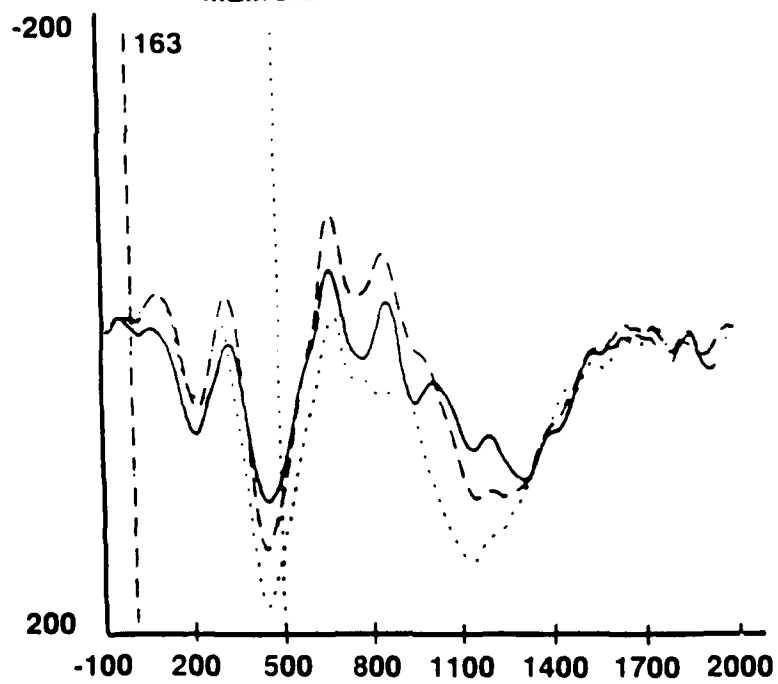


FIGURE 4 E

MEMSIZ 4 DELAY 15 NONTARGET



MEMSIZ 4 DELAY 4 NONTARGET

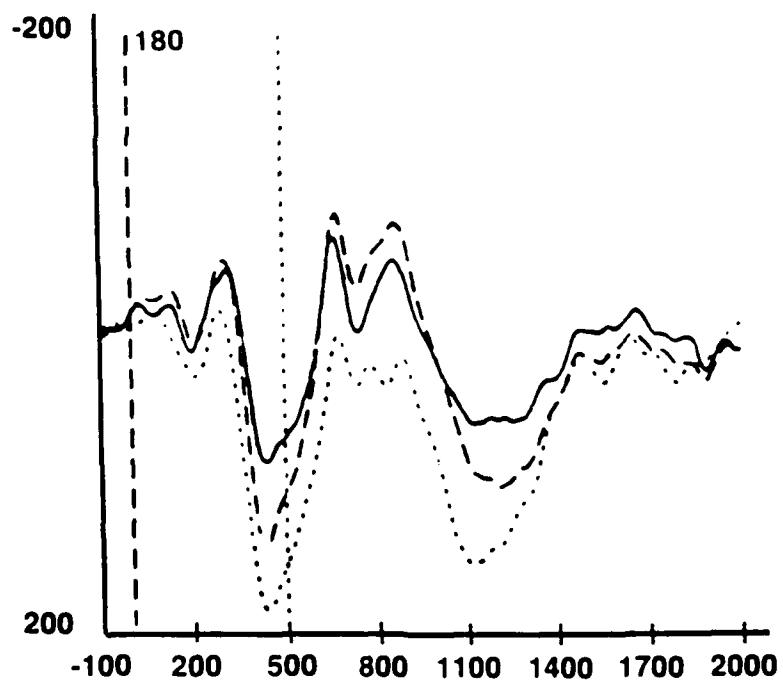


FIGURE 4 F

maximal correlation occurred was defined as the latency of the P300. The slope of the correlation between the template and the ERP was defined as the amplitude of the P300. The greater the slope, the larger the amplitude of the P300. Note that this procedure is unaffected by baseline shifts and is preferable to base to peak measures in this instance.

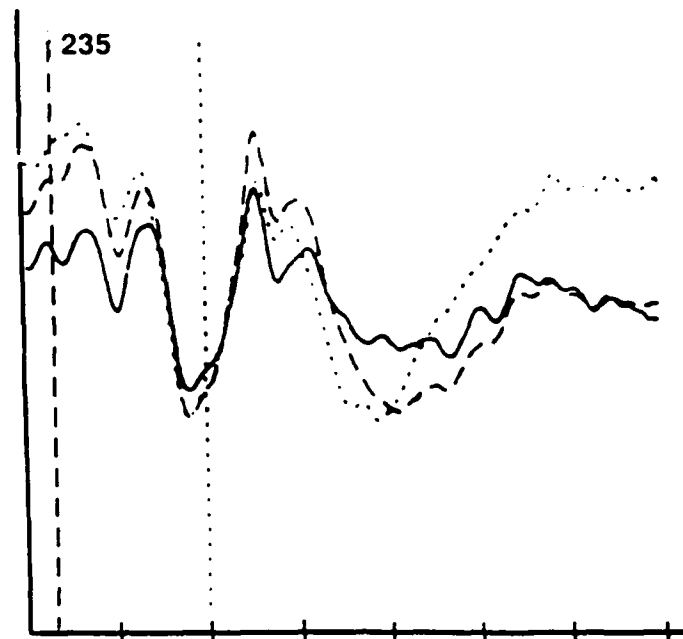
Examination of the latency of the P300s elicited by S1 revealed a significant response type X memory load X delay interaction ( $F(2,14)=11.5$ ,  $p<.01$ ,  $MSe=244$ ). However, the effects of delay were non-monotonic and there was no coherent pattern in the data. Because the interaction included the response type factor and the P300s were elicited prior to S2, this effect is probably unreliable. No other S1 P300 latency effects attained significance.

Examination of the amplitude of the S1 P300s revealed that P300 amplitude increased as a function of delay ( $F(1,7)=4.2$ ,  $p<.02$ ,  $MSe=229$ ). The effect was non-monotonic, with the smallest P300s elicited in the 0 delay condition. The difference between the 4 and 15 second delay conditions did not significantly differ ( $p<.05$ ). This increase in P300 amplitude in the 4 and 15 second delay conditions may reflect an increase in the allocation of processing resources as subjects switch from the distractor task to the Sternberg task. Additional research is required to fully interpret these differences.

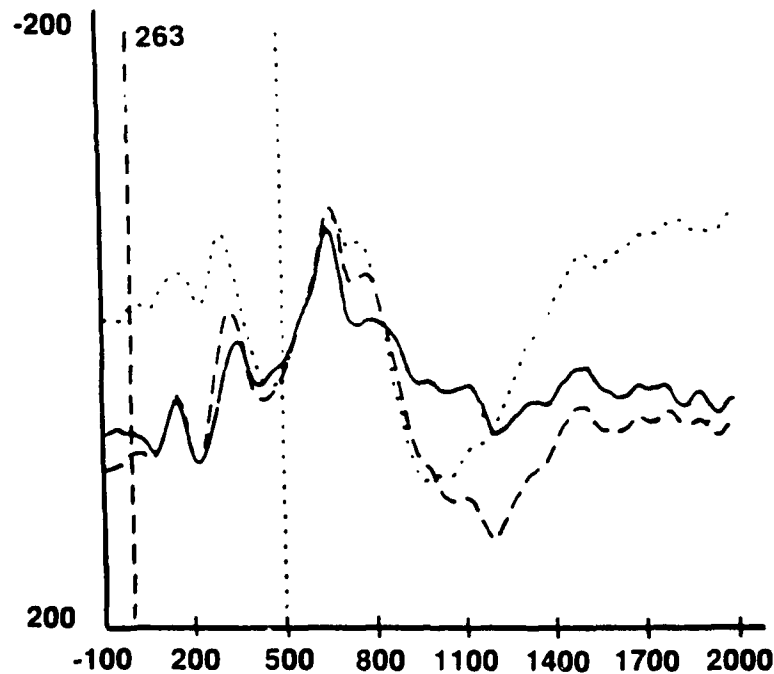
Examination of the P300s elicited by S2 provides the critical analysis of experiment 2. P300 latency increased as a function of memory load ( $F(1,7)=6.6$ ,  $p<.05$ ,  $MSe=4070$ ). The memory set size slope was 16.5 msec/item. P300 latency increased as a function of delay ( $F(2,14)=19.4$ ,  $p<.001$ ,  $MSe=1718$ ). The difference between the 0 and 15 second delay conditions was 59 msec. This effect of delay on P300 latency supports the hypothesis that the increase in reaction time from primary to secondary memory is due at least in part to the inclusion of a retrieval process inserted prior to the memory comparison process in secondary memory conditions. This interpretation is bolstered by the finding that P300 latency was sensitive to both memory load and delay, suggesting that the P300 was elicited after the memory comparison process. No other latency effects attained significance. In contrast to reaction time, P300 latency to targets was not significantly different from non-targets ( $F(1,7)=2.8$ ,  $p<.10$ ,  $MSe=5298$ ).

The P300 latency data suggest that the reaction time differences between targets and distractors is due to response-related processing. Furthermore, the effect of the delay was nearly twice as large with reaction time as with P300 latency. This suggests that delay may affect both stimulus evaluation and response-related processing. To evaluate the contribution of response-related processing, a comparison of the single trial RT/P300 ratio was conducted. By comparing the relative changes in reaction time and P300 latency, it is possible to localize the effects of different manipulations on information processing. Increases in response-related processing are identified as increases in the RT/P300 ratio. A ratio of 1.0 reflects the co-occurrence of reaction time and the peak of the P300. Ratios less than 1.0 indicate that the reaction time preceded peak P300 latency and ratios greater than 1.0 indicate that the peak P300 latency preceded the overt response. It is important to note that absolute RT/P300

# MEMSIZ 2 DELAY 4 TARGET



# MEMSIZ 2 DELAY 0 TARGET



Grand average scalp distribution ERPs from experiment 2. Ten subjects are included in the average. The baseline subtracted from each waveform is from S2 to 100 msec post-S2. The number of trials included in each average is displayed in the upper left-hand portion of each plot. The solid line represents the Fz electrode, dashed, Cz, and dotted, Pz. The experimental condition is presented above each panel. Figures a-f contain the 12 experimental conditions.

FIGURE 5 A

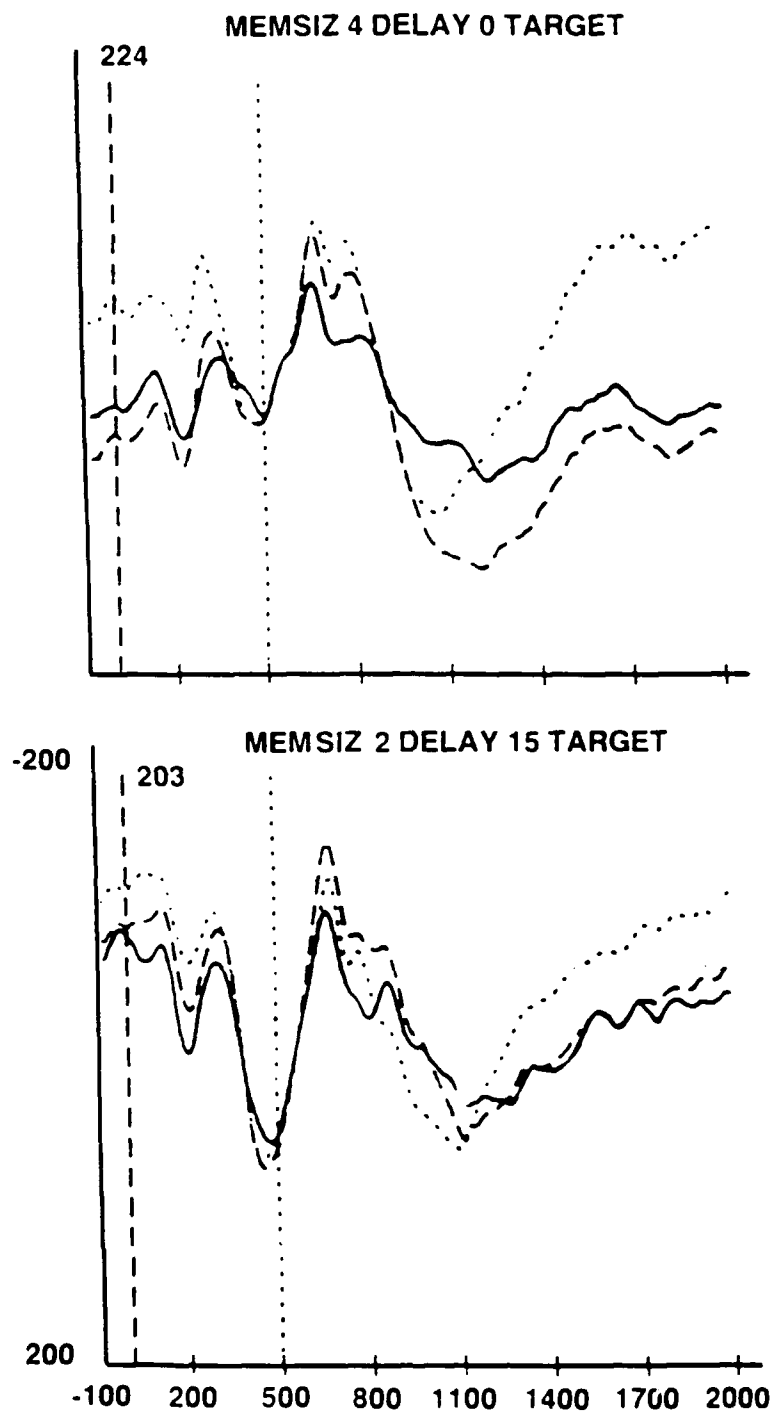
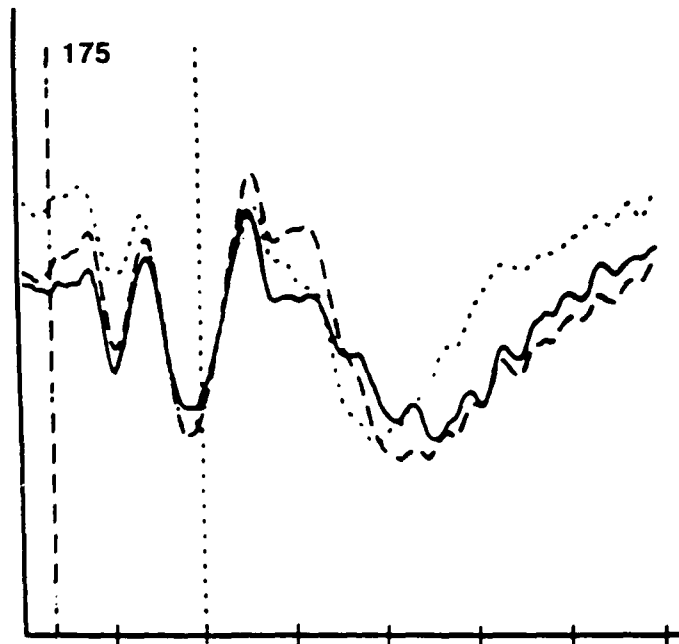


FIGURE 5 B

MEMSIZ 4 DELAY 15 TARGET



MEMSIZ 4 DELAY 4 TARGET

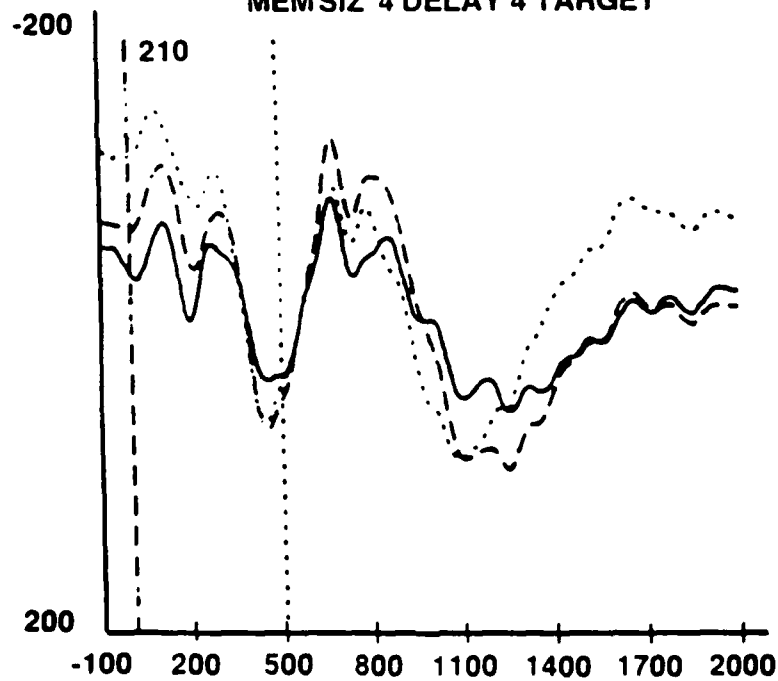
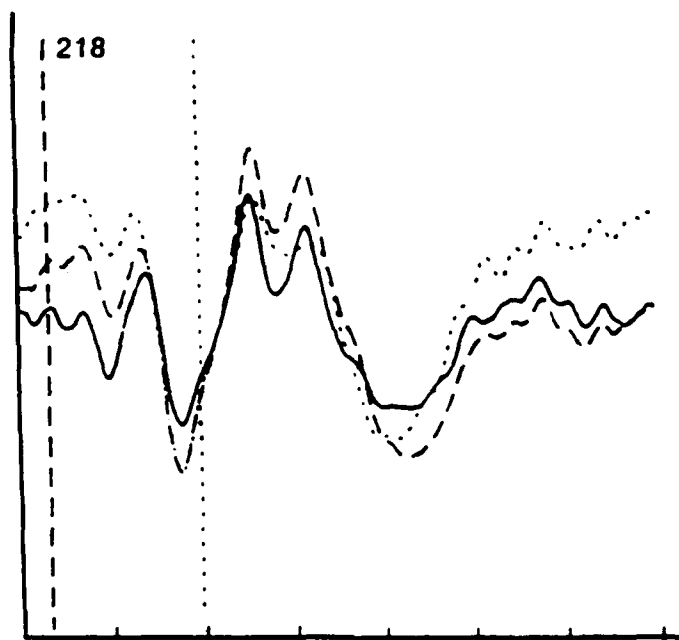


FIGURE 5 C

MEMSIZ 2 DELAY 4 NONTARGET



MEMSIZ 2 DELAY 0 NONTARGET

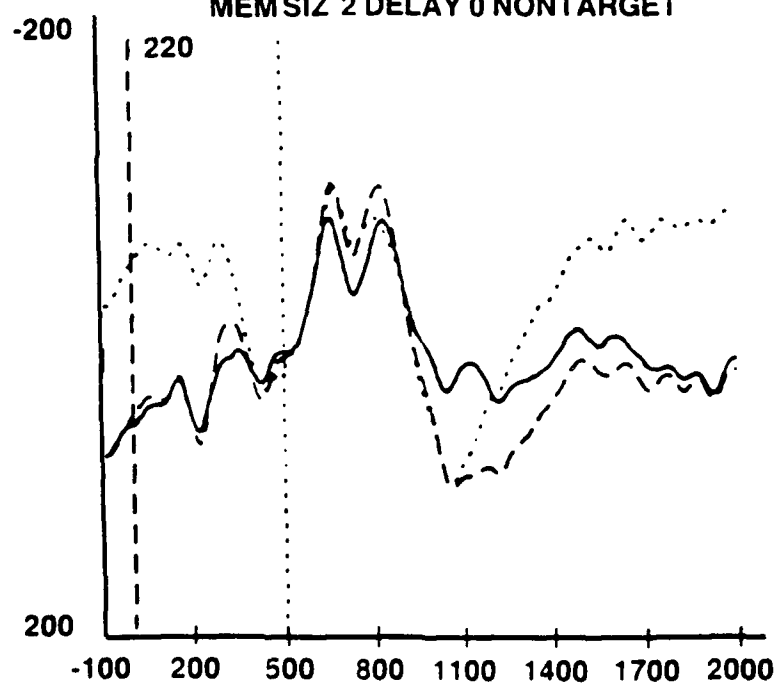
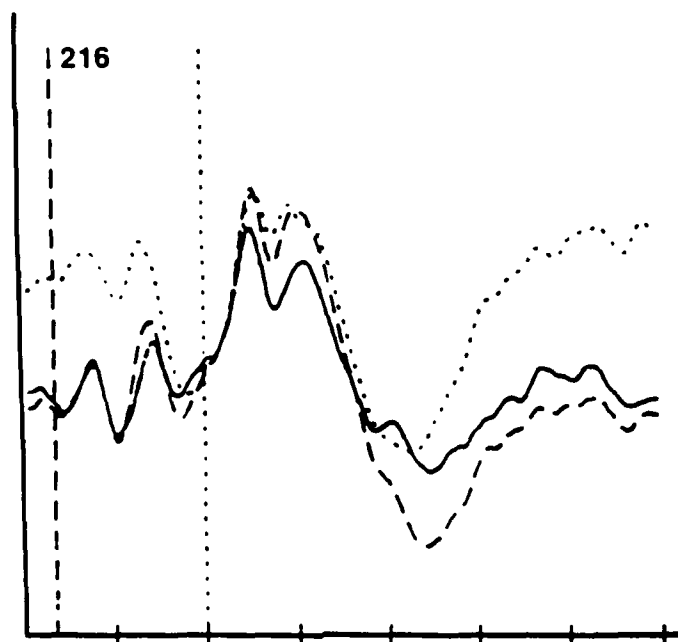
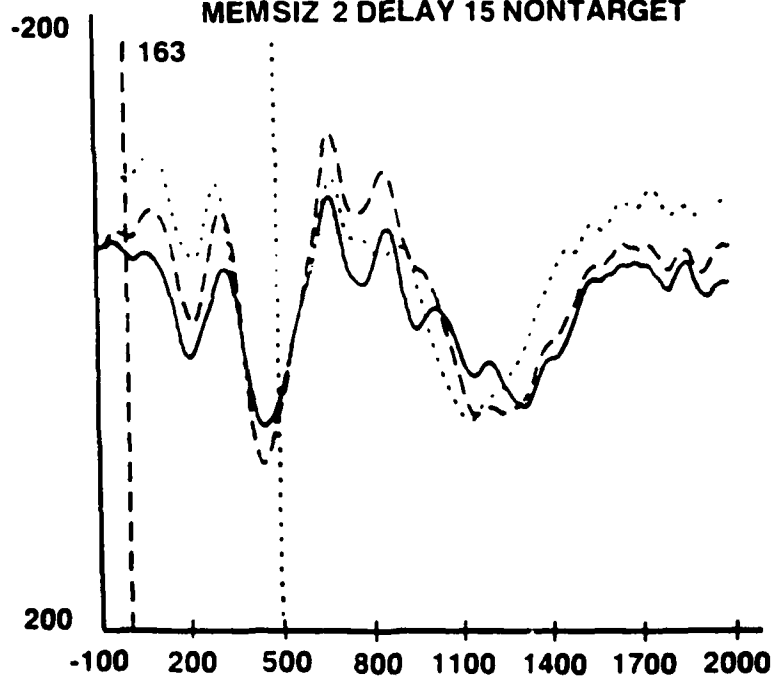


FIGURE 5 D

**MEMSIZ 4 DELAY 0 NONTARGET**



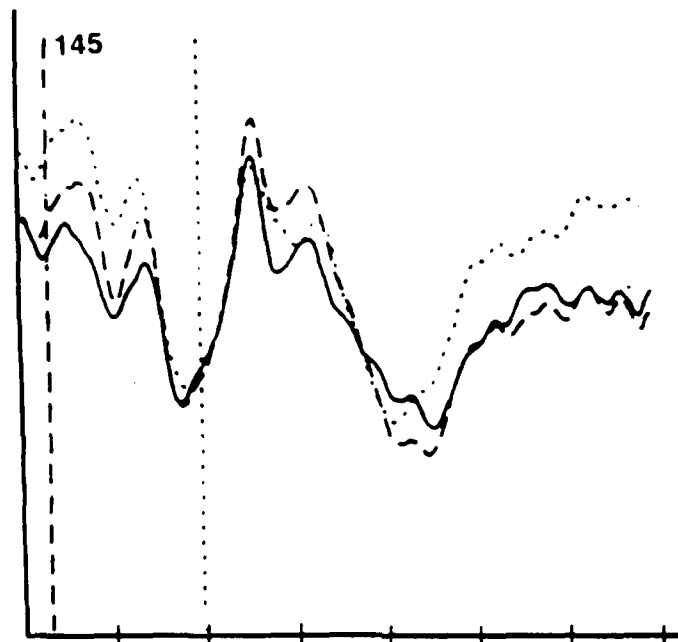
**MEMSIZ 2 DELAY 15 NONTARGET**



**FIGURE 5 E**



MEMSIZ 4 DELAY 15 NONTARGET



MEMSIZ 4 DELAY 4 NONTARGET

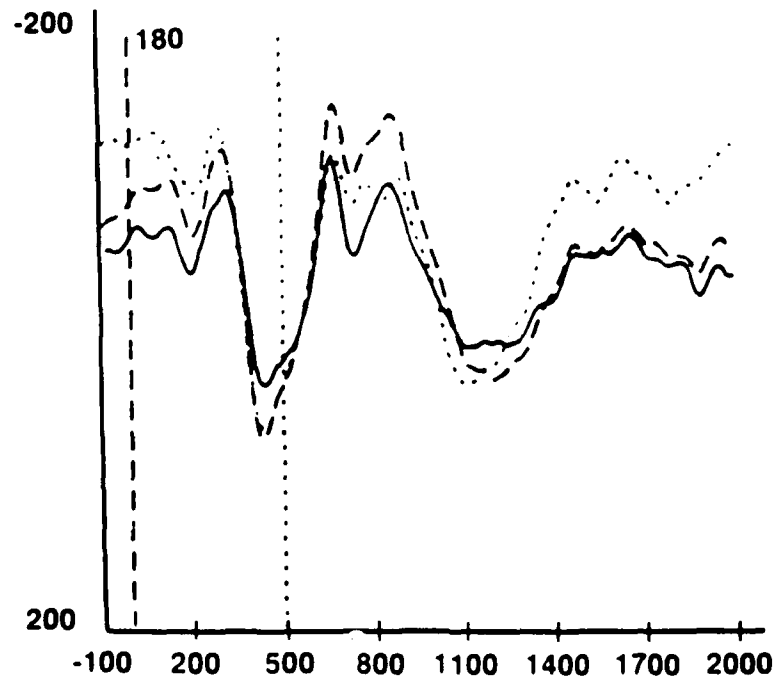
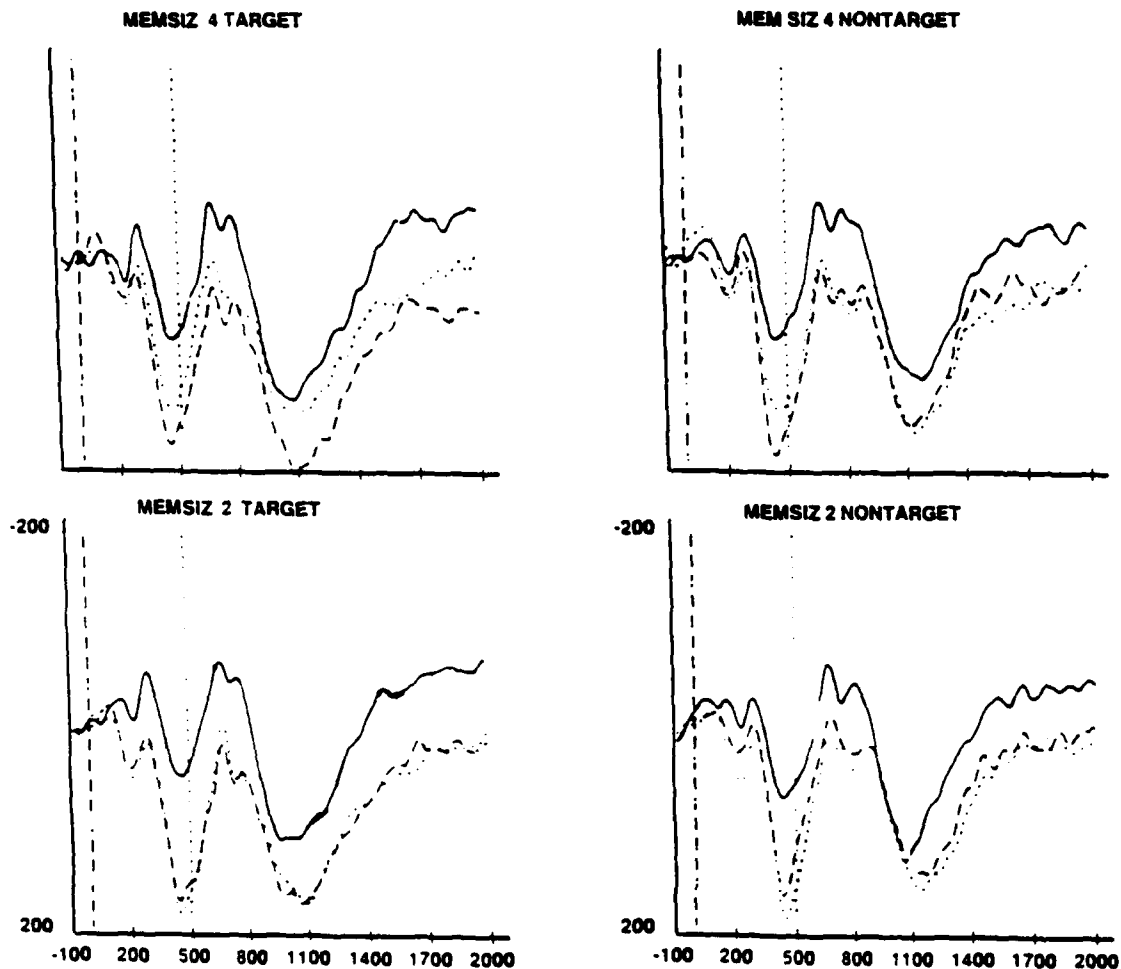


FIGURE 5 F



Grand average Pz overplots of delay. Ten subjects are included in the average. The baseline subtracted from each waveform is from 100 msec pre-S1 to S1. The solid line represents the 0 delay condition, the dashed line represents the 4 second delay condition, and the dotted line represents the 15 second delay condition. The experimental condition is described above each panel.

FIGURE 6

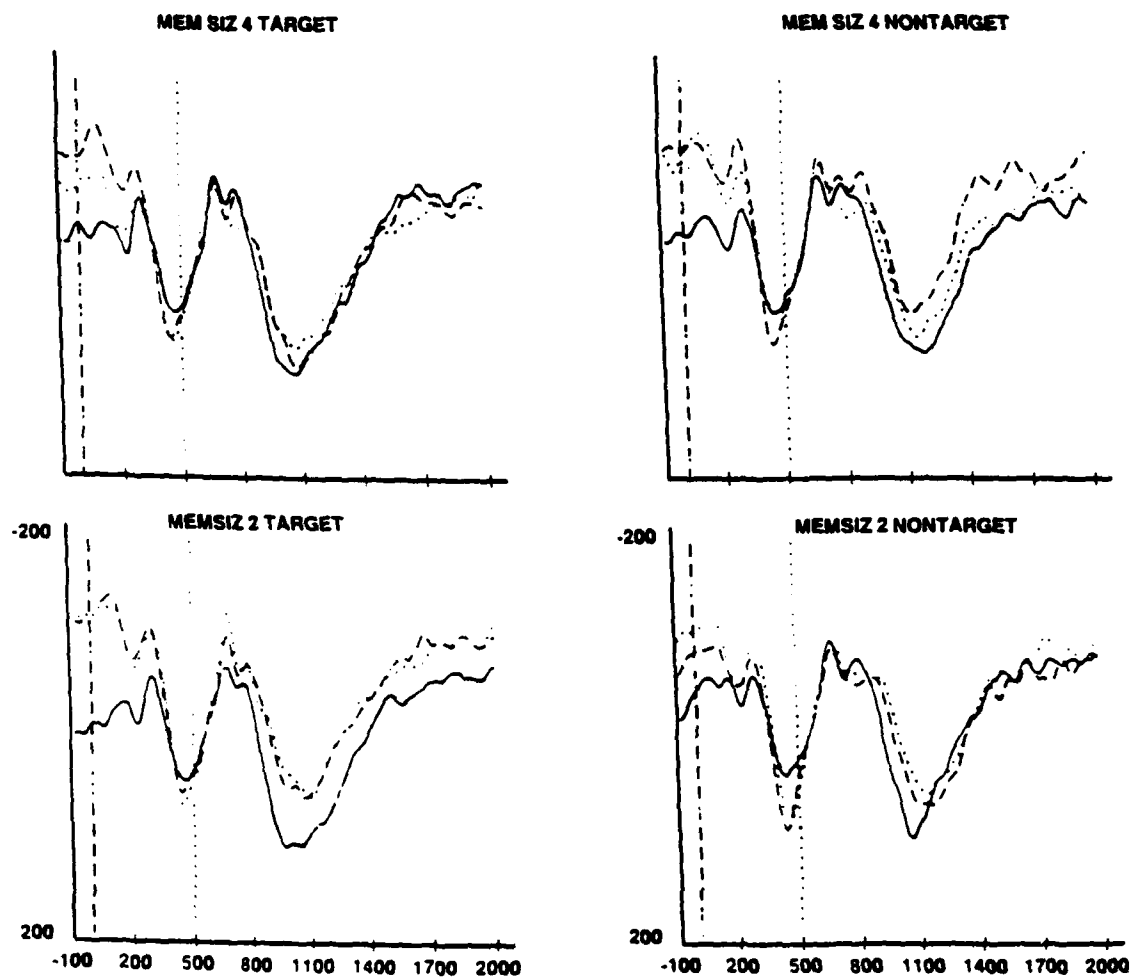
ratios are less meaningful than the relative differences between experimental conditions. This, in part, is due to the fact that we measure the latency of the peak of the P300 rather than the onset of the P300.

Examination of the single trial RT/P300 ratio revealed an increase from targets to non-targets ( $F(1,7)=17.3$ ,  $p<.01$ ,  $MSe=11554$ ), suggesting an increase in the response-related processing for non-targets. This increase in response-related processing could be due to response biases, uncertainty, rechecking, or some combination of these factors. The RT/P300 ratio increased as a function of delay ( $F(2,14)=5.3$ ,  $p<.01$ ,  $MSe=15266$ ). This is consistent with the hypothesis that some of the increase in reaction time as a function of delay is due to response-related processing. Finally, the RT/P300 ratio increased at a greater rate for non-targets than for targets ( $F(2,14)=7.3$ ,  $p<.006$ ,  $MSe=6569$ ). Post hoc comparisons revealed that this interaction was due to a large increase in the RT/P300 ratio for the non-target delay 15 condition and may be the result of uncertainty about the response, or to a rechecking.

Examination of the amplitude of the P300s elicited by S2 revealed a significant response type X delay interaction ( $F(2,14)=3.9$ ,  $p<.05$ ,  $MSe=101$ ). Post hoc analysis revealed that P300 amplitude was smaller for the 15 second delay condition for targets and larger for the 15 second delay condition for non-targets.

#### Conclusion; Experiment 2

Experiment 2 was conducted to examine the hypothesis that the increase in reaction time from primary to secondary memory was due to the insertion of a retrieval process prior to the memory comparison process, to response-related processing, or both. The evidence obtained using P300 latency as an index of stimulus evaluation processing suggests that both stimulus evaluation and response-related processing factors are affected by delay. These results suggest that the retrieval of information from secondary memory may be more rapid than the reaction time estimates suggest. The results also suggest that secondary memory includes a retrieval process inserted prior to the memory comparison process.



Grand average Pz overplots of delay. Ten subjects are included in the average. The baseline subtracted from each waveform is from S2 to 100 msec post-S2. The solid line represents the 0 delay condition, the dashed line represents the 4 second delay condition, and the dotted line represents the 15 second delay condition. The experimental condition is described above each panel.

FIGURE 7

## FOOTNOTES

Footnote 1. The terms primary and secondary memory originate with William James (1890, p. 646-648). Contemporary treatments can be found by Waugh and Norman (1965), Craik and Levey (1976), Atkinson and Shiffrin (1968), Baddeley and Hitch (1974), and Baddeley (1981). According to James, primary memory reflects the contents of consciousness, while secondary memory refers to information that no longer is in consciousness. Following Waugh and Norman (1965) primary memory can be defined in terms of its limited capacity and the rapid decay of information if rehearsal is not permitted. Rehearsal serves to maintain information in primary memory and transfer the information to secondary memory. Secondary memory is characterized as a large capacity storage system with a low forgetting rate.

Footnote 2. Care was employed selecting the words so as not to form any obvious categories and subjects were instructed not to use elaborative/chunking strategies when memorizing the memory set. These manipulations were intended to minimize any effectiveness of the compression of the memory set into subgroups of categories. Furthermore, since this was a VM task, it would be quite difficult for subjects to form associations for every pairwise and four-way comparison.

Footnote 3. Two subjects (subjects 8 and 9, see appendix N) were excluded from the analysis because of excessive error rate, resulting in a total sample size of 8.

## REFERENCES

- Anders, T. R., & Fozard, J. L. (1973). Effects of age upon retrieval from primary and secondary memory. Developmental Psychology, 9, 411-415.
- Atkinson, R. C., & Shiffrin, R. M. (1968). Human memory: A proposed system and its control processes. In K. W. Spence & J. T. Spence (Eds.), The Psychology of Learning and Motivation: Advances in Research and Theory. Vol. 2, (pp. 89-195). New York: Academic Press.
- Baddeley, A. (1981). The concept of working memory: A view of its current state and probable future development. Cognition, 10, 17-23.
- Baddeley, A. D., & Hitch, G. (1974). Working memory. In G. H. Bower (Ed.), The Psychology of Learning and Motivation. Vol. 8. New York: Academic Press.
- Cavanagh, J. P. (1972). Relation between the immediate memory span and memory search rate. Psychological Review, 79, 525-530.
- Clifton, C., Jr., & Birenbaum, S. (1970). Effects of serial position and delay of probe in a memory scan task. Journal of Experimental Psychology, 86, 69-76.
- Corballis, M. C., Kirby, J., & Miller, A. (1972). Access to elements of a memorized list. Journal of Experimental Psychology, 94, 185-190.
- Coyne, A. C., Allen, P. A., & Wickens, D. D. (1986). Influence of adult age on primary and secondary memory search. Psychology and Aging, 1, 187-194.
- Craik, F. I. M., & Levy, B. A. (1976). The concept of primary memory. In W. K. Estes (Ed.), Handbook of learning and cognitive processes. Vol. 4, (pp. 133-175). Hillsdale, NJ: Erlbaum.
- Donders, F. C. (1868) On the speed of mental processes. In: W. G. Koster (Ed. and trans.) Attention and Performance II. (pp. 412-431). Amsterdam: North-Holland (1969).
- Flexser, A. J. (1978). Long-term recognition latencies under rehearsal-controlled conditions: Do list-length effects depend on active memory? Journal of Experiment Psychology: Human Learning and Memory, 4, 47-54.
- Forrin, B., & Morin, R. E. (1969). Recognition times for items in short-and long-term memory. In W. G. Koster (Ed.), Acta Psychologica 30 Attention and Performance II. (pp. 126-141). Amsterdam: North-Holland Publishing Company.
- Gratton, G., Coles, M. G. H., & Donchin, E. (1983). A new method for off-line removal of ocular artifact. Electroencephalography and Clinical Neurophysiology, 55, 468-484.
- James, W. (1890). The Principles of Psychology. New York: Holt.

- Jasper, H. H. (1958). The ten-twenty electrode system of the International federation. Electroencephalography and Clinical Neurophysiology, 10, 371-375.
- Kucera, H., & Francis, W. N. (1967). Computational Analysis of Present Day English, Providence RI: Brown University Press.
- Magliero, A., Bashore, T. R., Coles, M. G. H., & Donchin, E. (1984). On the dependence of P300 latency on stimulus evaluation processes. Psychophysiology, 21, 171-186.
- McCarthy, G., & Donchin, E. (1981). A metric for thought: A comparison of P300 latency and reaction time. Science, 211, 77-80.
- Monsell, S. (1978). Recency, immediate recognition memory, and reaction time. Cognitive Psychology, 10, 465-501.
- Murdock, B. B., Jr. (1961). The retention of individual items. Journal of Experimental Psychology, 62, 618-625.
- Muter, P. (1980). Very rapid forgetting. Memory and Cognition, 8, 174-179.
- Peters, G. L. (1974). Coding processes in active and inactive memory. Journal of Experimental Psychology, 102, 423-430.
- Peterson, L. R. & Peterson, M. J. (1959). Short-term retention of individual verbal items. Journal of Experimental Psychology, 58, 193-198.
- Ratcliff, R. (1979). Group reaction time distributions and an analysis of distribution statistics. Psychological Bulletin, 86, 446-461.
- Sternberg, S. (1966). High-speed scanning in human memory. Science, 153, 652-654.
- Sternberg, S. (1969a). Memory-scanning: Mental processes revealed by reaction-time experiments. American Scientist, 57, 421-457.
- Sternberg, S. (1969b). The discovery of processing stages: Extensions of Donders' method. Acta Psychologica, 30, 276-315.
- Sternberg, S. (1975). Memory scanning: New findings and current controversies. Quarterly Journal of Experimental Psychology, 27, 1-32.
- Sternberg, S., Kroll, R. L., & Nasto, B. A. (1969). Retrieval from long-term vs. active memory. Paper presented at the meeting of the Psychonomics Society, St. Louis.
- Vincent, S. G. (1912). The function of the viborissae in the behavior of the white rat. Behavioral Monographs, 1, No. 5.
- Waugh, N. C., & Norman, D. A. (1965). Primary memory. Psychological Review, 72, 89-104.

Wickens, D. D., Moody, M. J., & Dow, R. (1981). The nature and timing of the retrieval process and of interference effects. Journal of Experimental Psychology: General, 110, 1-20.

Wickens, D. D., Moody, M. J., & Vidulich, M. (1985). Retrieval time as a function of memory set size, type of probes, and interference in recognition memory. Journal of Experimental Psychology: Learning, Memory, and Cognition, 11, 154-164.



## Appendix A

PMSM.PAS This is the Pascal source code for the primary/secondary memory experiment. The program is written in Turbo Pascal and assumes an enhanced graphics adapter (EGA) card. The program should be compiled prior to initial execution. Two INCLUDE files are included at compile time: CRT.PAS and TIMER.PAS. The source for these procedures can be found in appendix B and C, respectively. The inclusion of these procedures is accomplished by the {\$I CRT.PAS} and {\$I TIMER.PAS} commands at the beginning of the program.

A description of the experimental procedure can be found in the methods section of the paper. Instructions to subjects are included as appendix M. To run the program an input stimulus file and an output data file must be specified e.g., Type: PMSM <input file> <output file>. The program will not proceed unless the stimulus input file exists on disc and the output data file does not exist on disc. A sample stimulus input file is included as appendix F. This file contains 20 word stimuli, the response assignment keys for 'Yes' and 'No' responses, and 96 primary/secondary memory trial blocks. The program presents the 96 trial blocks (run time approximately 20 minutes) and then writes 108 identification parameters for each block to the output file. Program PMSM pauses prior to the first block and after every 16 trial blocks to provide rest breaks. Subjects continue the program by pressing the enter key. Several options may be specified by pressing designated keys during run time:

'Q' will quit the program and write the output to disc.

'P' will pause the program. To continue, press enter.

'F' will provide trial by trial feedback to subjects. Once this option is specified, it remains in effect throughout the 96 trial blocks.

{ Modification date: Jan 12, 1988  
Programmer: David Strayer  
University of Illinois

Program to control primary/secondary memory study

ids output by the program:

id(1) - subject number  
id(2) - session  
id(3) - block  
id(4) - delay  
id(5) - soa of retrieval cue  
id(6) - size of memory set  
id(7) - constant (1) or varied (2) mapping  
id(8) - ASCII code for Yes\_button  
id(9) - ASCII code for No\_button

Running memory ids

Trial 1

id(10) - running memory stimulus  
id(11) - rt (for running memory trial)  
id(12) - stim type for running memory (1=target, 2=nontarget)  
id(13) - response for running memory (0=none, 1=target, 2=nontarget)  
id(14) - correctness for running memory: 0=no response, 1=hit,  
          2=correct rejection, 3=false alarm, 4=miss

Trial 2

id(15) - running memory stimulus  
id(16) - rt (for running memory trial)  
id(17) - stim type for running memory (1=target, 2=nontarget)  
id(18) - response for running memory (0=none, 1=target, 2=nontarget)  
id(19) - correctness for running memory: 0=no response, 1=hit,  
          2=correct rejection, 3=false alarm, 4=miss

Trial 3

id(20) - running memory stimulus  
id(21) - rt (for running memory trial)  
id(22) - stim type for running memory (1=target, 2=nontarget)  
id(23) - response for running memory (0=none, 1=target, 2=nontarget)  
id(24) - correctness for running memory: 0=no response, 1=hit,  
          2=correct rejection, 3=false alarm, 4=miss

Trial 4

id(25) - running memory stimulus  
id(26) - rt (for running memory trial)

id(27) - stim type for running memory (1=target, 2=nontarget)  
id(28) - response for running memory (0=none, 1=target, 2=nontarget)  
id(29) - correctness for running memory: 0=no response, 1=hit,  
2=correct rejection, 3=false alarm, 4=miss

Trial 5

id(30) - running memory stimulus  
id(31) - rt (for running memory trial)  
id(32) - stim type for running memory (1=target, 2=nontarget)  
id(33) - response for running memory (0=none, 1=target, 2=nontarget)  
id(34) - correctness for running memory: 0=no response, 1=hit,  
2=correct rejection, 3=false alarm, 4=miss

Trial 6

id(35) - running memory stimulus  
id(36) - rt (for running memory trial)  
id(37) - stim type for running memory (1=target, 2=nontarget)  
id(38) - response for running memory (0=none, 1=target, 2=nontarget)  
id(39) - correctness for running memory: 0=no response, 1=hit,  
2=correct rejection, 3=false alarm, 4=miss

Trial 7

id(40) - running memory stimulus  
id(41) - rt (for running memory trial)  
id(42) - stim type for running memory (1=target, 2=nontarget)  
id(43) - response for running memory (0=none, 1=target, 2=nontarget)  
id(44) - correctness for running memory: 0=no response, 1=hit,  
2=correct rejection, 3=false alarm, 4=miss

Trial 8

id(45) - running memory stimulus  
id(46) - rt (for running memory trial)  
id(47) - stim type for running memory (1=target, 2=nontarget)  
id(48) - response for running memory (0=none, 1=target, 2=nontarget)  
id(49) - correctness for running memory: 0=no response, 1=hit,  
2=correct rejection, 3=false alarm, 4=miss

Trial 9

id(50) - running memory stimulus  
id(51) - rt (for running memory trial)  
id(52) - stim type for running memory (1=target, 2=nontarget)  
id(53) - response for running memory (0=none, 1=target, 2=nontarget)  
id(54) - correctness for running memory: 0=no response, 1=hit,  
2=correct rejection, 3=false alarm, 4=miss

Trial 10

id(55) - running memory stimulus  
id(56) - rt (for running memory trial)  
id(57) - stim type for running memory (1=target, 2=nontarget)  
id(58) - response for running memory (0=none, 1=target, 2=nontarget)  
id(59) - correctness for running memory: 0=no response, 1=hit,  
2=correct rejection, 3=false alarm, 4=miss

Trial 11

id(60) - running memory stimulus  
id(61) - rt (for running memory trial)  
id(62) - stim type for running memory (1=target, 2=nontarget)  
id(63) - response for running memory (0=none, 1=target, 2=nontarget)  
id(64) - correctness for running memory: 0=no response, 1=hit,  
2=correct rejection, 3=false alarm, 4=miss

Trial 12

id(65) - running memory stimulus  
id(66) - rt (for running memory trial)  
id(67) - stim type for running memory (1=target, 2=nontarget)  
id(68) - response for running memory (0=none, 1=target, 2=nontarget)  
id(69) - correctness for running memory: 0=no response, 1=hit,  
2=correct rejection, 3=false alarm, 4=miss

Trial 13

id(70) - running memory stimulus  
id(71) - rt (for running memory trial)  
id(72) - stim type for running memory (1=target, 2=nontarget)  
id(73) - response for running memory (0=none, 1=target, 2=nontarget)  
id(74) - correctness for running memory: 0=no response, 1=hit,  
2=correct rejection, 3=false alarm, 4=miss

Trial 14

id(75) - running memory stimulus  
id(76) - rt (for running memory trial)  
id(77) - stim type for running memory (1=target, 2=nontarget)  
id(78) - response for running memory (0=none, 1=target, 2=nontarget)  
id(79) - correctness for running memory: 0=no response, 1=hit,  
2=correct rejection, 3=false alarm, 4=miss

Trial 15

id(80) - running memory stimulus  
id(81) - rt (for running memory trial)  
id(82) - stim type for running memory (1=target, 2=nontarget)  
id(83) - response for running memory (0=none, 1=target, 2=nontarget)  
id(84) - correctness for running memory: 0=no response, 1=hit,



```

Program pmsm ;
{$C-}      {disable cntrl c option}
{$I TIMER.PAS} {include timer routines}
{$I CRT.PAS} {CRT control routines (synch and cursor manipulation)}

```

```

var

```

```

    ID : array[1..96,1..108] of integer;
    Condition_codes : array[1..96,1..7] of integer;

```

```

    CM_targ: array[1..5,1..4] of char;
    CM_dist: array[1..5,1..4] of char;
    VM_stimuli: array[1..10,1..4] of char;
    Targ: array[1..5,1..4] of char;
    Dist: array[1..5,1..4] of char;

```

```

    stmdur,    {duration of stimulus}
    nback,     {number back in running memory task}
    block,     {block number}
    subject,   {subject number}
    session,   {session number}
    soa,       {stimulus onset asynchrony}
    cmvm,      {consistent (1) or varied (2) mapping}
    memsiz,    {memory set size}
    stype,     {stim type (pos or neg) for Stern trial}
    delay: integer; { delay from memory set to probe}

```

```

    Yes_button: char; {key to press for positive response}
    No_button:  char; {key to press for negative response}

```

```

    s : TimeStamp;    {Timer variable for TimerMilli calls}

```

```

    Numfile,      {true if number of files is 2}
    Newfile,      {true if output file does not exist on disc}
    Exists,       {true if input file exists on disc}
    scrnon,       {true if stim on, false if stim off}
    pause,        {true if P key pressed}
    fback,        {true if F key pressed}
    done : Boolean; {true if Q key pressed}

```

```

    Infile,
    Outfile: Text;

```

```
Function RandomTwixt(low,high: Integer): Integer ;  
  
{ Procedure to return a random integer between the values of low and high }  
begin  
RandomTwixt := Random(high-low+1)+low;  
end;
```

```
Procedure Init_all; { intializes variables }
```

```
var
```

```
i,
```

```
j: integer;
```

```
begin
```

```
  CrtCursorOff; {turn off the cursor}  
  clrscr;       {clear the screen}  
  stmdur := 200; {stimulus duration}  
  nback  := 2;   {number back in r-memory}  
  block  := 0;   {block counter}  
  TimerInit;    {initializes the clock}  
  done := false; {true if Q key pressed}  
  fback := false; {true if F key pressed}  
  pause := true; {true if P key pressed}
```

```
  for i := 1 to 96 do
```

```
    for j := 1 to 108 do
```

```
      begin
```

```
        id[i,j] := 0;          {zero id array}
```

```
      end;
```

```
end;
```



```

Procedure Read_stim;

var
  i,
  j: integer;

begin

  for i := 1 to 5 do      {read in CM target stimuli}
  begin
    readln(Infile,CM_targ[i]);
  end;

  for i := 1 to 5 do      {read in CM distractor stimuli}
  begin
    readln(Infile,CM_dist[i]);
  end;

  for i := 1 to 10 do     {read in VM stimuli}
  begin
    readln(Infile,VM_stimuli[i]);
  end;

  readln(Infile,Yes_button,No_button); {read key assignments}

  for j:= 1 to 96 do      {read in condition codes}

  begin
    for i:= 1 to 7 do read(Infile,condition_codes[j,i]);
    if j < 96 then readln(Infile);
  end;

end; {read_stim}

```

```
Procedure Stim_Sel; {shuffle stimuli and choose targ and dist stimuli}
```

```
var
```

```
    templ : char;  
    loop, {loop counter}  
    i,  
    j,  
    k : integer; {variable equal to a randomly selected array position}
```

```
begin
```

```
    block := block+1;           {increment block counter}  
    if block = 96 then done := true; {exit program after this block}  
    if (block mod 16) = 0 then pause := true;
```

```
    subject := condition_codes[block,1];  
    session := condition_codes[block,2];  
    soa      := condition_codes[block,3];  
    delay    := condition_codes[block,4];  
    cmvm     := condition_codes[block,5];  
    memsiz   := condition_codes[block,6];  
    stype    := condition_codes[block,7];
```

```
{store id values}
```

```
    id[block,1] := subject;  
    id[block,2] := session;  
    id[block,3] := block;  
    id[block,4] := delay;  
    id[block,5] := soa;  
    id[block,6] := memsiz;  
    id[block,7] := cmvm;  
    id[block,8] := ord(Yes_button);  
    id[block,9] := ord(No_button);
```

```
for k := 1 to 10 do
```

```
begin
```

```
for loop := 1 to 5 do {permute CM targets}
```

```
begin
```

```
    j := RandomTwixt(1,5);  
    for i := 1 to 4 do  
        begin  
            templ := CM_targ[j,i];  
            CM_targ[j,i] := CM_targ[loop,i];  
            CM_targ[loop,i] := templ;
```

```
        end; {for}
```

```
end; {for}
```

```

for loop := 1 to 5 do {permute CM distractors}
begin

    j := RandomTwixt(1,5);
    for i := 1 to 4 do
        begin
            templ := CM_dist[j,i];
            CM_dist[j,i] := CM_dist[loop,i];
            CM_dist[loop,i] := templ;
        end; {for}
    end; {for}

for loop := 1 to 10 do {permute VM array}
begin

    j := RandomTwixt(1,10);
    for i := 1 to 4 do
        begin
            templ := VM_stimuli[j,i];
            VM_stimuli[j,i] := VM_stimuli[loop,i];
            VM_stimuli[loop,i] := templ;
        end; {for}
    end; {for}

end; {for}

    if cmvm = 1 then {choose CM stimuli}
        begin

            for j := 1 to memsiz do
                for i := 1 to 4 do
                    begin
                        targ[j,i] := CM_targ[j,i]; {target array}
                    end;

                for j := 1 to 5 do
                    for i := 1 to 4 do
                        begin
                            dist[j,i] := CM_dist[j,i]; {distractor array}
                        end;
                    end;
                end;

            end;

            if cmvm = 2 then {choose VM stimuli}
                begin

                    for j := 1 to memsiz do
                        for i := 1 to 4 do
                            begin
                                targ[j,i] := VM_stimuli[j,i]; {target array}
                            end;
                        end;
                    end;
                end;
            end;
        end;
    end;

```

```
    for j := 1+memsiz to 5+memsiz do
    for i := 1 to 4 do
        begin
            dist[j-memsiz,i] := VM_stimuli[j,i]; {distractor array}
        end;
    end;

end;

end; {stimulus select}
```

```

Procedure Memory_Set;      {procedure to present Memory Set}

var
j,
i: integer;
word: string[4];
response: char;
begin

    if keypressed then      {clear the Kbd buffer}
    begin
        read(kbd,response);
        if upcase(response) = 'Q' then done := true;
        if upcase(response) = 'F' then fback := true;
        if upcase(response) = 'P' then pause := true;
    end; {if/then}

    for i := 1 to memsiz do
        begin                {presentation of memory set}

            if memsiz = 2 then gotoXY(27+(i*8),8);    {position cursor}
            if memsiz = 4 then gotoXY(19+(i*8),8);    {position cursor}
            word := targ[i,1]+targ[i,2]+targ[i,3]+targ[i,4];
            writeln(trm,word);    {turn on memory set}
            for j :=1 to 4 do id[block,80+(4*i)+j] :=ord(targ[i,j]);

        end;

        TimerStamp(s);      {start the clock}

        while TimerMilli(s) < 3000 do;
            clrscr;          {turn off memory set}

            while TimerMilli(s) <4500 do;

        end; {memory set procedure}

```

```

Procedure Run_Trial;      {procedure to present Run trial }

var

    RT,
    pos_neg,
    itr1,
    jstim: integer;

    lstim: array[1..3] of integer;

    response : char;

begin

if delay <> 0 then {if delay is 0, bypass running memory trials}

begin

for itr1:= 1 to Abs(delay) do
begin
    if keypressed then      {clear the Kbd buffer}
    begin
        read(kbd,response);
        if upcase(response) = 'Q' then done := true;
        if upcase(response) = 'F' then fback := true;
        if upcase(response) = 'P' then pause := true;
    end; {if/then}

        RT := -1;          {set RT = -1}

        TimerStamp(s);      {start the clock}

{ prepare stimuli during pre-stimulus interval (5 msec)}

        pos_neg := RandomTwixt(1,3);
        if pos_neg = 3 then pos_neg := 2;

{
    if a positive trial, repeat stimulus nback
    if a negative trial, select a new stimulus}

        if pos_neg = 1 then {positive trial}
        begin
            if itr1 <= nback then jstim := RandomTwixt(1,9);
            if itr1 > nback then jstim := lstim[nback];
        end;
        if pos_neg = 2 then {negative trial}
        begin
            jstim := RandomTwixt(1,9);

```

```

        while jstim = lstim[nback] do
        begin
            jstim := RandomTwixt(1,9);
        end;

    end; {if/then}

{store successive stimuli}

        lstim[3] := lstim[2];
        lstim[2] := lstim[1];
        lstim[1] := jstim;
        id[block,(itr1*5)+5] := jstim;

    while TimerMilli(s) < 5 do;          {wait out prestimulus interval}

        gotoXY(40,11);                  {position cursor}
        CrtSyncEnd;                      {sync CRT}
        TimerStamp(s);                   {start the clock}
        if delay > 0 then
        writeln(trm,jstim);              {turn on stimulus}
        scrnon := true;                  {set screen flag}

        while TimerMilli(s) < 1000 do

            begin
            if timerMilli(s) >= stmdur then
            if scrnon then
                begin
                clrscr;                   {turn screen off}
                scrnon := false;          {reset screen flag}
            end;

            if RT = -1 then                {if no response has been made}

                if keypressed then        {check for keypress}
                begin
                    RT := TimerMilli(s);  {set RT = elapsed time}
                    read(kbd,response);    {read which key was
pressed}

                end; {if/then}

            end;

        { classify trial and store RT and stim type (1=hit, 2=cr, 3=fa, 4=miss)}

```

```

id[block,(itrl*5)+6] := rt;
id[block,(itrl*5)+7] := pos_neg;
id[block,(itrl*5)+8] := 0;
id[block,(itrl*5)+9] := 0;

if rt > 0 then
    begin
        if upcase(response) = Yes_button then
            begin
                id[block,(itrl*5)+8] := 1;
                if id[block,(itrl*5)+7] = 1 then id[block,(itrl*5)+9] := 1;
                if id[block,(itrl*5)+7] = 2 then id[block,(itrl*5)+9] := 3;
            end;

            if upcase(response) = No_button then
                begin
                    id[block,(itrl*5)+8] := 2;
                    if id[block,(itrl*5)+7] = 2 then id[block,(itrl*5)+9] := 2;
                    if id[block,(itrl*5)+7] = 1 then id[block,(itrl*5)+9] := 4;
                end;
        end; {if/then}
    end; {for}
end; {if/then}
end; {running memory procedure}

```



```

Procedure Stern_Trial;    {procedure to present Stern trial }

var
    RT,
    j,
    i: integer;
    word: string[4];

    response: char;
begin
    if keypressed then      {clear the Kbd buffer}
    begin
        read(kbd,response);
        if upcase(response) = 'Q' then done := true;
        if upcase(response) = 'F' then fback := true;
        if upcase(response) = 'P' then pause := true;
    end; {if/then}

    RT := -1;                {set RT = -1}

    { deliver retrieval cue if appropriate }

    if soa > 0 then
    begin
        gotoXY(40,11);      {set position on screen}
        CrtSyncEnd;          {sync CRT}
        TimerStamp(s);        {start the clock}
        writeln(trm,'*');    {turn on retrieval cue}
        gotoXY(40,11);      {set position on screen}

        while TimerMilli(s) < stmdur do; {wait for end of stmdur}
        writeln(trm,' ');    {turn off retrieval cue}

        while TimerMilli(s) < soa do;

    end; {if/then}

    { prepare stimuli during pre-stimulus interval (5 msec)}

    TimerStamp(s);          {start the clock}
    gotoXY(39,13);          {set position on screen}

    if stype = 1 then {positive trial}
    begin
        i := RandomTwixt(1,memsiz);
        word := targ[i,1]+targ[i,2]+targ[i,3]+targ[i,4];
        for j :=1 to 4 do id[block,104+j] :=ord(targ[i,j]);
    end;
    if stype = 2 then {negative trial}

```

```

begin
  i := RandomTwixt(1,5);
  word := dist[i,1]+dist[i,2]+dist[i,3]+dist[i,4];
  for j :=1 to 4 do id[block,104+j] :=ord(dist[i,j]);
end;

while TimerMilli(s) < 5 do;      {wait out prestimulus interval}

CrtSyncEnd;                    {sync CRT}
TimerStamp(s);                 {start the clock}
writeln(trm,word);             {turn on stimulus}
scrnon := true;                {set screen flag}

while TimerMilli(s) < 3000 do

  begin
    if timerMilli(s) >= stmdur then
      if scrnon then
        begin
          clrscr;                {turn screen off}
          scrnon := false;       {reset screen flag}
        end;

        if RT = -1 then          {if no response has been made}

          if keypressed then     {check for keypress}
            begin
              RT := TimerMilli(s); {set RT = elapsed time}
              read(kbd,response);  {read which key was
pressed}

            end; {if/then}

          end;

        { classify trial and store RT and stim type (1=hit, 2=cr, 3=fa, 4=miss)}

        id[block,101] := rt;
        id[block,102] := stype;
        id[block,103] := 0;
        id[block,104] := 0;

        if rt > 0 then
          begin

```

```

    if upcase(response) = Yes_button then
        begin
            id[block,103] := 1;
            if id[block,102] = 1 then id[block,104] := 1;
            if id[block,102] = 2 then id[block,104] := 3;
        end;

    if upcase(response) = No_button then
        begin
            id[block,103] := 2;
            if id[block,102] = 2 then id[block,104] := 2;
            if id[block,102] = 1 then id[block,104] := 4;
        end;

    end; {if/then}

end; {sternberg procedure}

```

```

Procedure Pause_program; {procedure to pause program}

begin

    if pause then
    begin
        clrscr;
        gotoXY(20,20);
        if block = 0 then write(trm,'      Hit ENTER to initiate program      ');
        if block > 0 then write(trm,'Program paused: Hit ENTER to continue');
        while not keypressed do;
            clrscr;
            pause := false;
        end;
    end;
end;

```

```

Procedure Feed_back;      {procedure to present feedback}

var
    error,
    corr,
    i: integer;

begin
    error := 0;
    corr := 0;

    if fback then
        begin

        if delay <> 0 then
            begin
                for i := 3 to Abs(delay) do
                    begin
                        case(id[block,(i*5)+9]) of
                            0,3,4: error := error+1;
                            1,2: corr := corr+1;
                        end;
                    end;
                end;

                gotoXY(25,23);
                write(trm,' Run -- ',corr,' correct trials out of ',corr+error);
            end;

            gotoXY(25,24);
            case (id[block,104]) of
                0: write (trm,' Stern -- No response detected');
                1,2: write (trm,' Stern -- Correct response ');
                3,4: write (trm,' Stern -- Incorrect response ');
            end;
        end;
    end;
end;

```

```

Procedure Session_end;    {operations following session termination}

var
  lines,
  i,
  j: integer;

begin
  timerkill;  {kill the clock}

  for i := 1 to 96 do
    begin
      lines := 1;
      for j := 1 to 108 do
        begin
          write(Outfile,id[i,j]:5);  {write data to disc}
          if(lines mod 16) = 0 then writeln(Outfile);
          lines := lines+1;
        end;
        writeln(Outfile);
      end;

      CrtCursorOn;                      {turn the cursor back on}
      Close (Infile);
      Close (Outfile);
    end;
  end;
end;

```

```

Begin {Main program loop}

  Numfile := false;

  if ParamCount <> 2 then
    begin
      writeln(trm,'ERROR: Incorrect number of parameters');
      writeln(trm);
      writeln(trm,'Correct format: pmsm <stimulus file> <output file>');
    end;

  if ParamCount = 2 then
    begin
      Numfile := true;

      Assign(Infile,ParamStr(1));
      {$I-} Reset(Infile) {$I+};
      Exists := (IOresult = 0);
      if Not Exists then
        writeln(trm,'ERROR: Input file does not exist');

      if Exists then
        begin
          Assign(Outfile,ParamStr(2));
          {$I-} Reset(Outfile) {$I+};
          Newfile := (IOresult <> 0);
          if Not Newfile then
            writeln(trm,'ERROR: Output file already exists');
          if Newfile then
            Rewrite(Outfile);
        end;

    end;

  If Numfile and Exists and Newfile then
    begin
      Init_All;           { initialize counters and loop variables}
      Read_Stim;          { read in the stimuli and conditions from disc}

    repeat
      Pause_program;      { holds program until key is struck}
      Stim_Sel;           { selects targ and dist stimuli}
      Memory_Set;         { present Memory set }
      Run_Trial;          { present Run trial }
      Stern_Trial;        { present Stern trial }
      Feed_back;          { present feedback if option set}

    until done;

  Session_end;           {Write trials to disc and terminate session}
end;

```

END. {pmsm.pas}



## Appendix B

CRT.PAS This is the include file for CRT routines which permit synchronization of the display and control of the cursor. The file is automatically included with the {\$I CRT.PAS} command in program PMSM.PAS.

```
procedure CrtModule ; external 'CRT.BIN' ;  
procedure CrtSyncEnd ; external CrtModule[$3] ;  
procedure CrtSyncStart ; external CrtModule[$6] ;  
procedure CrtCursorOn ; external CrtModule[$9] ;  
procedure CrtCursorOff ; external CrtModule[$C] ;
```

## Appendix C

TIMER.PAS This is the timer procedure routines included with the {\$I  
TIMER.PAS} command in program PMSM.PAS. This code enables a clock used for  
millisecond timing of events.

```

type
  TimeStamp =
    record
      count,low,high: Integer
    end ;

procedure TimerModule; external 'TIMER.BIN';
procedure TimerInit ; external TimerModule[$3] ;
procedure TimerKill ; external TimerModule[$6] ;
procedure TimerStamp(var stamp: TimeStamp) ; external TimerModule[$9] ;
function TimerMilli(stamp: TimeStamp): Integer ; external TimerModule[$C] ;

function UnsignedIntegerToReal(u: Integer): Real ;
var
  x: Real ;

begin { UnsignedIntegerToReal }
  if u>=0 then x := u
  else x := u+65536.0 ;
  UnsignedIntegerToReal := x ;
end { UnsignedIntegerToReal } ;

function TimeStampToReal(stamp: TimeStamp): Real ;
const
  ShiftLeft16 = 65536.0;

var
  rLow,rHigh,rCount: Real ;
  total: Real ;

begin { TimeStampToReal }
  rCount := UnsignedIntegerToReal(stamp.count) ;
  rLow := UnsignedIntegerToReal(stamp.low) ;
  rHigh := UnsignedIntegerToReal(stamp.high) ;
  total := rHigh ;
  total := total*ShiftLeft16+rLow ;
  total := total*ShiftLeft16+rCount ;
  TimeStampToReal := total ;
end { TimeStampToReal } ;

procedure TimeOfDay(var hour,min,sec,milli: Integer) ;
const
  TimeBase = 14.31818e+06 ;
  milliDiv = 1.193180e+03 ;
  secDiv = 1.193180e+06 ;
  minDiv = 7.159080e+07 ;
  hourDiv = 4.295448e+09 ;
  ShiftLeft16 = 65536.0 ;

var
  stamp: TimeStamp ;

```

```
total: Real ;

begin { TimeOfDay }
  TimerStamp(stamp) ;
  total := TimeStampToReal(stamp) ;
  hour := Trunc(total/hourDiv) ;
  total := total-total*hourDiv ;
  min := Trunc(total/minDiv) ;
  total := total-total*minDiv ;
  sec := Trunc(total/secDiv) ;
  total := total-total*secDiv ;
  milli := Round(total/milliDiv)
end { TimeOfDay } ;
```

## Appendix D

STMGEN.PAS This turbo Pascal program generates the stimulus input files used by program PMSM. The program reads 20 four letter words from file STM.DAT (see appendix E), then prompts the user for the 'Yes' and 'No' Key assignments (which are to be entered on the same line without separation, e.g., Z/ would specify the 'Yes' key is Z and the 'No' key is /). The program will generate 96 trial blocks. Appendix F illustrates the output of STMGEN. The files generated by this program can also be used by program PMSM.FLX, the fortran version of PMSM which collects ERP data (see appendix I).

```

{ Modification date: Jan 16, 1988}

Program Stmgen;

Function RandomTwixt(low,high: Integer): Integer ;

{ Procedure to return a random integer between the values of low and high }
begin
RandomTwixt := Random(high-low+1)+low;
end;

var
  i,
  j,
  k,
  loop,
  temp,
  itrial,
  rep,
  soa,
  cmvm,
  delay,
  memsiz,
  stype,
  session,
  subject: integer;

  templ,
  yes_button,
  no_button: char;

  stimuli: array[1..20,1..4] of char;
  trial : array[1..96,1..7] of integer;
  block : array[1..96] of integer;

  sub: string[2];
  ses: string[2];

  infile : text;
  outfile : text;

begin

write('Enter subject number ');
read(subject);
writeln;
write('Enter keys for yes and no ');
read(yes_button,no_button);
writeln;

```

```

{assign stimulus input file}

assign(infile,'stm.dat');
reset(infile);

{read words from disc}

for i := 1 to 20 do      {read in stimuli}
  begin
    readln(infile,stimuli[i]);
  end;

{close words file}
close(infile);

for session := 1 to 12 do
  begin
    itrial := 0;
    cmvm := 2;
    soa := 500;

    for rep := 1 to 8 do
      for delay := 1 to 3 do
        for memsiz := 1 to 2 do
          for stype := 1 to 2 do
            begin
              itrial := itrial+1;
              block[itrial] := itrial;
              trial[itrial,1] := subject;
              trial[itrial,2] := session;
              trial[itrial,3] := soa;

              if delay = 1 then trial[itrial,4] := 0;
              if delay = 2 then trial[itrial,4] := 4;
              if delay = 3 then trial[itrial,4] := 15;

              trial[itrial,5] := cmvm;
              trial[itrial,6] := memsiz*2;
              trial[itrial,7] := stype;
            end;

          {permute condition}

          for k := 1 to 40 do
            for loop := 1 to 96 do
              begin
                j := RandomTwixt(1,96);
                temp := block[j];
                block[j] := block[loop];
                block[loop] := temp;
              end;
            end;
          end;
        end;
      end;
    end;
  end;
end;

```



```

end;

{assign disc file}
str(subject,sub);
str(session,ses);
writeln(trm,'Generating file: Stm',sub,'_',ses,'.dat');
assign(outfile,'stm'+sub+'_'+ses+'.dat');
rewrite(outfile);

{write stimulus files to disc}

for i := 1 to 20 do
  begin
    writeln(outfile,stimuli[i]);
  end;

writeln(outfile,yes_button,no_button);

for i := 1 to 96 do
  begin
    for j := 1 to 7 do write(outfile,trial[block[i],j]:5);
    writeln(outfile);
  end;
close(outfile);
end;
end.

```

## Appendix E

File STM.DAT Input file used by program STMGEN to generate stimulus sequence files for program PMSM. The file contains 20 four letter words.

SIGN  
CLAY  
WISH  
GAME  
FARM  
DEAL  
HOUR  
CLUB  
HAIR  
SHIP  
FILE  
RAIN  
DUST  
TONE  
EDGE  
TERM  
MARK  
BANK  
ROSE  
EASE

## Appendix F

File STM1-1.DAT Stimulus input file used by program PMSM. This file is the output of program STMGEN. This file contains 20 word stimuli, the response assignment keys for 'Yes' and 'No' responses, and 96 primary/secondary memory trial parameters. Each of the 96 trials has seven parameters: They are: subject, session, stimulus onset asynchrony (SOA) between the retrieval cue and the probe, number of running memory trials between the memory set and the probe (Delay), consistent (1) or varied mapping of stimulus to response, memory set size, and stimulus response type (target=1, non-target=2).

SIGN  
CLAY  
WISH  
GAME  
FARM  
DEAL  
HOUR  
CLUB  
HAIR  
SHIP  
FILE  
RAIN  
DUST  
TONE  
EDGE  
TERM  
MARK  
BANK  
ROSE  
BASE  
Z/

1	1	500	4	2	2	1
1	1	500	0	2	2	1
1	1	500	15	2	4	2
1	1	500	15	2	4	2
1	1	500	4	2	4	2
1	1	500	4	2	2	2
1	1	500	15	2	2	1
1	1	500	0	2	4	2
1	1	500	15	2	4	2
1	1	500	15	2	2	1
1	1	500	4	2	4	1
1	1	500	4	2	4	1
1	1	500	0	2	2	1
1	1	500	0	2	2	1
1	1	500	0	2	4	2
1	1	500	0	2	2	2
1	1	500	15	2	4	1
1	1	500	0	2	4	2
1	1	500	0	2	4	2
1	1	500	4	2	2	2
1	1	500	0	2	2	2
1	1	500	4	2	4	1
1	1	500	4	2	2	1
1	1	500	4	2	2	1
1	1	500	0	2	4	1
1	1	500	4	2	2	1
1	1	500	15	2	4	1
1	1	500	4	2	2	1
1	1	500	4	2	4	1
1	1	500	0	2	2	1

1	1	500	15	2	4	1
1	1	500	4	2	2	1
1	1	500	15	2	2	1
1	1	500	0	2	2	1
1	1	500	15	2	2	1
1	1	500	15	2	4	1
1	1	500	0	2	2	2
1	1	500	0	2	2	1
1	1	500	4	2	4	1
1	1	500	15	2	2	2
1	1	500	0	2	4	1
1	1	500	4	2	4	2
1	1	500	15	2	4	1
1	1	500	0	2	4	1
1	1	500	4	2	2	2
1	1	500	4	2	2	2
1	1	500	15	2	2	2
1	1	500	15	2	2	2
1	1	500	15	2	2	2
1	1	500	15	2	4	2
1	1	500	15	2	2	1
1	1	500	0	2	4	2
1	1	500	0	2	4	1
1	1	500	4	2	4	2
1	1	500	15	2	4	2
1	1	500	15	2	2	1
1	1	500	0	2	4	1
1	1	500	0	2	2	2
1	1	500	4	2	2	2
1	1	500	15	2	2	2
1	1	500	0	2	2	2
1	1	500	0	2	4	2
1	1	500	4	2	2	1
1	1	500	15	2	2	1
1	1	500	4	2	4	2
1	1	500	0	2	4	1
1	1	500	4	2	2	1
1	1	500	15	2	4	2
1	1	500	15	2	4	1
1	1	500	15	2	4	1
1	1	500	4	2	4	1
1	1	500	15	2	2	1
1	1	500	4	2	4	1
1	1	500	0	2	4	1
1	1	500	0	2	2	2
1	1	500	15	2	4	1
1	1	500	15	2	4	2
1	1	500	0	2	2	1
1	1	500	0	2	4	2
1	1	500	15	2	2	2
1	1	500	15	2	2	2

1	1	500	4	2	4	2
1	1	500	4	2	2	2
i	1	500	0	2	4	1
1	1	500	0	2	2	1
1	1	500	4	2	2	2
1	1	500	4	2	4	2
1	1	500	0	2	2	2
1	1	500	4	2	4	2
1	1	500	4	2	4	2
1	1	500	15	2	4	2
1	1	500	4	2	4	1
1	1	500	15	2	2	2
1	1	500	0	2	4	2
1	1	500	4	2	2	2
1	1	500	0	2	2	2

## Appendix G

AVE.PAS Program to examine output file created by PMSM and give quick summary statistics. This program will need to be customized for the experimental conditions.



```

{$g512}      {redirect input from disc}
{$p512}      {redirect output to disc}
program ave;

{Modification date: Mar 20, 1987}

var
  id : array[1..108] of integer;
  ave : array [1..2,1..2,1..2,1..3,1..3,1..2] of real;
  rmem : array [1..2,1..2,1..3,1..4] of real;
  subject,
  session,
  lines,
  delay,
  soa,
  memsiz,
  cmvm,
  posneg,
  corr,
  point,
  i,
  j: integer;

  rt: real;

begin
  {blank output array}

  for cmvm := 1 to 2 do
  for posneg := 1 to 2 do
  for memsiz := 1 to 2 do
  for soa := 1 to 3 do
  for delay := 1 to 3 do
    begin
      ave[cmvm,posneg,memsiz,soa,delay,1] := 0.0;
      ave[cmvm,posneg,memsiz,soa,delay,2] := 0.0;
      rmem[cmvm,memsiz,delay,1] := 0.0;
      rmem[cmvm,memsiz,delay,2] := 0.0;
      rmem[cmvm,memsiz,delay,3] := 0.0;
      rmem[cmvm,memsiz,delay,4] := 0.0;
    end;

  {loop through data file — increment appropriate bins}

  while not EOF do
  begin
    lines :=1;
    for j := 1 to 108 do
      begin
        read(id[j]);

```

```

if(lines mod 16) =0 then readln;
lines := lines+1;
end;
readln;

```

```

{assign values read from disc}

```

```

if id[4] = 0 then delay := 1;      {1 = delay 0}
if id[4] = 4 then delay := 2;      {2 = delay 4}
if id[4] = 15 then delay := 3;     {3 = delay 15}

```

```

if id[5] = 200 then soa := 1;      {1 = soa 200}
if id[5] = 500 then soa := 2;      {2 = soa 500}
if id[5] = 1000 then soa := 3;     {3 = soa 1000}

```

```

posneg := id[102];                {1=pos, 2=neg}
memsiz := id[6] div 2;             {1=memsiz2, 2=memsiz4}
cmvm := id[7];                     {1=cm, 2=vm}
rt := id[101];                     {reaction time}
corr := id[104];                   {correctness: 1=hit, 2=cr, 3=fa, 4=miss}
subject := id[1];                  {subject number}
session := id[2];                  {session number}

```

```

case(corr) of

```

```

1,2:

```

```

begin
ave[cmvm,posneg,memsiz,soa,delay,1] :=
ave[cmvm,posneg,memsiz,soa,delay,1]+rt;
ave[cmvm,posneg,memsiz,soa,delay,2] :=
ave[cmvm,posneg,memsiz,soa,delay,2]+1.0;
end;

```

```

end;

```

```

for i := 3 to id[4] do

```

```

begin
rt := id[(i*5)+6];
case(id[(i*5)+9]) of
1,2: begin
rmem[cmvm,memsiz,delay,1] :=
rmem[cmvm,memsiz,delay,1]+1.0;

rmem[cmvm,memsiz,delay,4] :=
rmem[cmvm,memsiz,delay,4]+rt;
end;

```

```

3,4: rmem[cmvm,memsiz,delay,2] :=
rmem[cmvm,memsiz,delay,2]+1.0;

```

```

0 : rmem[cmvm,memsiz,delay,3] :=
rmem[cmvm,memsiz,delay,3]+1.0;

```

```

end;

```

```

end;

end;

{calculate averages and write to output file}

writeln('Subject ',subject);
writeln('Session ',session);

for cmvm := 1 to 2 do
begin
if cmvm = 1 then writeln('Consistent Mapping');
if cmvm = 2 then writeln('Varied Mapping');
for posneg := 1 to 2 do
begin
if posneg = 1 then writeln('Positive trials');
if posneg = 2 then writeln('Negaitve trials');
writeln;
writeln('Delay          0          4          15  ');
for memsiz := 1 to 2 do
for soa := 1 to 3 do
begin
if memsiz = 1 then write('M = 2: SOA = ',soa:1);
if memsiz = 2 then write('M = 4: SOA = ',soa:1);
for delay := 1 to 3 do
begin
if ave[cmvm,posneg,memsiz,soa,delay,2] > 0 then
ave[cmvm,posneg,memsiz,soa,delay,1] :=
ave[cmvm,posneg,memsiz,soa,delay,1]/
ave[cmvm,posneg,memsiz,soa,delay,2];
write(ave[cmvm,posneg,memsiz,soa,delay,1]:5:0);
write(ave[cmvm,posneg,memsiz,soa,delay,2]:4:0);
end;
writeln;
end;
writeln;
end;
writeln;
writeln('Running memory performance');
writeln;
write('Memsiz Delay #corr #err #nrsp    RT');
writeln;
for memsiz := 1 to 2 do
for delay := 2 to 3 do
begin
if rmem[cmvm,memsiz,delay,1] > 0 then
rmem[cmvm,memsiz,delay,4] :=
rmem[cmvm,memsiz,delay,4]/
rmem[cmvm,memsiz,delay,1];

rmem[cmvm,memsiz,delay,4] := rmem[cmvm,memsiz,delay,4];

```

```
write((memsiz*2):6);  
if delay = 2 then write('    4');  
if delay = 3 then write('    15');  
for point := 1 to 4 do write(rmem[cmvm,memsiz,delay,point]:6:0);  
writeln;  
  
end;  
end;  
end.
```

## Appendix H

PMSM.FLX Source code for primary/secondary memory experiment used to collect ERPs. The file is written using a flecs preprocessor. See appendix I for the Fortran Source Code.

The experimental procedure is identical to the Pascal version of the program. After compilation, to execute the program type "RU PMSM". The program prompts the user for a stimulus input file (which should be in identical format to appendix F). A menu will be presented with the following options:

(R)un (T)ape (C)alib (Q)uit. "R" initiates the experiment. "Q" Quits the program. "C" gathers 5 calibration trials. "T" enters the tape manipulation menu. The tape menu consists of the following options: (E)xit the menu and return to main menu, (R)ewind the tape, (L)abel the tape header, (S)kip a specified number of EOFs, (V)erify the data written to tape, (W)rite an EOF.

Two special real-time options are provided: If the user types "P" the program will pause until the user presses carriage return. If the user types "Q", the run-time program will abort and return to the main menu.

The program assumes 5 channels of data are to be recorded. The channels are 1) EOG, 2) Fz, 3) Cz, 4) Pz, and 5) stimulus marker. Each trial block is written to tape with 108 identification parameters and 5 multiplexed data channels.

```

C
C   PMSM.FLX
C
C   MODIFICATION DATE: JAN 16, 1988
C
C   PROGRAM TO CONTROL PRIMARY/SECONDARY MEMORY STUDY
C
C   IDS OUTPUT BY THE PROGRAM:
C
C   ID(1)  - SUBJECT NUMBER
C   ID(2)  - SESSION
C   ID(3)  - BLOCK
C   ID(4)  - DELAY
C   ID(5)  - SOA OF RETRIEVAL CUE
C   ID(6)  - SIZE OF MEMORY SET
C   ID(7)  - CONSTANT (1) OR VARIED (2) MAPPING
C   ID(8)  - CODE FOR YES BUTTON
C   ID(9)  - CODE FOR NO BUTTON
C
C   RUNNING MEMORY IDS
C
C   TRIAL 1
C
C   ID(10) - RUNNING MEMORY STIMULUS
C   ID(11) - RT (FOR RUNNING MEMORY TRIAL)
C   ID(12) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)
C   ID(13) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)
C   ID(14) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,
C           2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS
C
C   TRIAL 2
C
C   ID(15) - RUNNING MEMORY STIMULUS
C   ID(16) - RT (FOR RUNNING MEMORY TRIAL)
C   ID(17) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)
C   ID(18) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)
C   ID(19) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,
C           2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS
C
C   TRIAL 3
C
C   ID(20) - RUNNING MEMORY STIMULUS
C   ID(21) - RT (FOR RUNNING MEMORY TRIAL)
C   ID(22) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)
C   ID(23) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)
C   ID(24) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,
C           2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS
C
C   TRIAL 4
C
C   ID(25) - RUNNING MEMORY STIMULUS

```

C ID(26) - RT (FOR RUNNING MEMORY TRIAL)  
 C ID(27) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)  
 C ID(28) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)  
 C ID(29) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,  
 C 2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS  
 C  
 C TRIAL 5  
 C  
 C ID(30) - RUNNING MEMORY STIMULUS  
 C ID(31) - RT (FOR RUNNING MEMORY TRIAL)  
 C ID(32) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)  
 C ID(33) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)  
 C ID(34) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,  
 C 2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS  
 C  
 C TRIAL 6  
 C  
 C ID(35) - RUNNING MEMORY STIMULUS  
 C ID(36) - RT (FOR RUNNING MEMORY TRIAL)  
 C ID(37) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)  
 C ID(38) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)  
 C ID(39) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,  
 C 2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS  
 C  
 C TRIAL 7  
 C  
 C ID(40) - RUNNING MEMORY STIMULUS  
 C ID(41) - RT (FOR RUNNING MEMORY TRIAL)  
 C ID(42) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)  
 C ID(43) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)  
 C ID(44) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,  
 C 2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS  
 C  
 C TRIAL 8  
 C  
 C ID(45) - RUNNING MEMORY STIMULUS  
 C ID(46) - RT (FOR RUNNING MEMORY TRIAL)  
 C ID(47) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)  
 C ID(48) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)  
 C ID(49) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,  
 C 2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS  
 C  
 C TRIAL 9  
 C  
 C ID(50) - RUNNING MEMORY STIMULUS  
 C ID(51) - RT (FOR RUNNING MEMORY TRIAL)  
 C ID(52) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)  
 C ID(53) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)  
 C ID(54) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,  
 C 2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS  
 C

```

C   TRIAL 10
C
C   ID(55) - RUNNING MEMORY STIMULUS
C   ID(56) - RT (FOR RUNNING MEMORY TRIAL)
C   ID(57) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)
C   ID(58) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)
C   ID(59) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,
C           2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS
C
C   TRIAL 11
C
C   ID(60) - RUNNING MEMORY STIMULUS
C   ID(61) - RT (FOR RUNNING MEMORY TRIAL)
C   ID(62) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)
C   ID(63) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)
C   ID(64) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,
C           2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS
C
C   TRIAL 12
C
C   ID(65) - RUNNING MEMORY STIMULUS
C   ID(66) - RT (FOR RUNNING MEMORY TRIAL)
C   ID(67) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)
C   ID(68) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)
C   ID(69) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,
C           2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS
C
C   TRIAL 13
C
C   ID(70) - RUNNING MEMORY STIMULUS
C   ID(71) - RT (FOR RUNNING MEMORY TRIAL)
C   ID(72) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)
C   ID(73) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)
C   ID(74) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,
C           2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS
C
C   TRIAL 14
C
C   ID(75) - RUNNING MEMORY STIMULUS
C   ID(76) - RT (FOR RUNNING MEMORY TRIAL)
C   ID(77) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)
C   ID(78) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)
C   ID(79) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,
C           2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS
C
C   TRIAL 15
C
C   ID(80) - RUNNING MEMORY STIMULUS
C   ID(81) - RT (FOR RUNNING MEMORY TRIAL)
C   ID(82) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)
C   ID(83) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)

```



```

C   ID(84) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,
C           2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS
C
C   STERNBERG IDS
C
C   ID(85) - MEMORY SET WORD #1, LETTER #1
C   ID(86) - MEMORY SET WORD #1, LETTER #2
C   ID(87) - MEMORY SET WORD #1, LETTER #3
C   ID(88) - MEMORY SET WORD #1, LETTER #4
C
C   ID(89) - MEMORY SET WORD #2, LETTER #1
C   ID(90) - MEMORY SET WORD #2, LETTER #2
C   ID(91) - MEMORY SET WORD #2, LETTER #3
C   ID(92) - MEMORY SET WORD #2, LETTER #4
C
C   ID(93) - MEMORY SET WORD #3, LETTER #1
C   ID(94) - MEMORY SET WORD #3, LETTER #2
C   ID(95) - MEMORY SET WORD #3, LETTER #3
C   ID(96) - MEMORY SET WORD #3, LETTER #4
C
C   ID(97) - MEMORY SET WORD #4, LETTER #1
C   ID(98) - MEMORY SET WORD #4, LETTER #2
C   ID(99) - MEMORY SET WORD #4, LETTER #3
C   ID(100) - MEMORY SET WORD #4, LETTER #4
C
C   ID(101) - RT (FOR STERNBERG TRIAL)
C   ID(102) - STIM TYPE (1=TARGET, 2=NONTARGET)
C   ID(103) - RESPONSE (0=NONE, 1=TARGET, 2=NONTARGET)
C   ID(104) - CORRECTNESS: 0=NO RESPONSE, 1=HIT, 2=CORRECT REJECTION,
C           3=FALSE ALARM, 4=MISS
C
C   ID(105) - STIMULUS PRESENTED, LETTER #1
C   ID(106) - STIMULUS PRESENTED, LETTER #2
C   ID(107) - STIMULUS PRESENTED, LETTER #3
C   ID(108) - STIMULUS PRESENTED, LETTER #4
C
C   -----
C
C   IMPLICIT INTEGER (A-Z)
C   INTEGER COND(96,7),LSTIM(3),ISEED(2)
C   BYTE NAME(17),ERR,TEMP1,WORD(5),NUM(9),DIGIT(2),YES,NO,
C   +CMTARG(5,4),CMDIST(5,4),VMSTIM(10,4),TARG(5,4),DIST(5,4)
C   REAL SEED,RAN
C   EQUIVALENCE (SEED,ISEED),(ISEED(1),S1),(ISEED(2),S2)
C   COMMON /RAN/ S1,S2
C   COMMON /DATA/ ID(108),DATA(5,210)
C
C   DATA STMDUR,NBACK/200,2/
C   DATA NCHAN,NPTS,DRATE,NIDS,BASE/5,210,10,108,100/
C   DATA NUM/'1','2','3','4','5','6','7','8','9'/

```

```

C
C   GET A RANDOM SEED AND WARM UP RANDOM NUMBER GENERATOR
C
  SEED=SECNDS(0)
  DO (RANDOM=1,2000)
  CALL RAN(S1,S2)
  FIN

C
C   INITIALIZE MATROX
C
  CALL MCINIT(1)

C
C   SET UP DISPLAY ASSIGNMENTS
C
  CALL GQINIT
  CALL GQTTL(' PM/SM EXPERIMENT')
  CALL GQLABL(1,'FZ ELECTRODE')
  CALL GQLABL(2,'CZ ELECTRODE')
  CALL GQLABL(3,'PZ ELECTRODE')
  CALL GQLABL(4,'EOG ELECTRODE')
  CALL GQLABL(5,'STM MARKER')

C
C   ASSIGN STIMULUS FILE FROM DISC
C
  TYPE *,'INPUT STIMULUS FILE FOR THIS SUBJECT'
  CALL GETSTR(5,NAME,16,ERR)
  CALL ASSIGN(40,NAME,0,'OLD')

C
C   READ STIMULUS FILE
C
  DO (I=1,5)
  READ(40,1) (CMTARG(I,J),J=1,4)
  FIN

C
  DO (I=1,5)
  READ(40,1) (CMDIST(I,J),J=1,4)
  FIN

C
  DO (I=1,10)
  READ(40,1) (VMSTIM(I,J),J=1,4)
  FIN

C
  READ(40,2) YES,NO
  IF(YES.EQ.'/')
  YESBUT=1
  NOBUT=0
  FIN
  IF(YES.EQ.'Z')
  YESBUT=0
  NOBUT=1
  FIN

```

```

C
    DO (J=1,96)
    READ(40,3) (COND(J,I),I=1,7)
    FIN

C
1    FORMAT(4A1)
2    FORMAT(2A1)
3    FORMAT(7I5)
C
C ===== GET THE OPTION FROM THE USER =====
C
C    GET THE OPTION
C
100   CONTINUE
    TYPE 110
110   FORMAT(// '$ MENU: (R)UN, (T)APE, (C)ALIB, (Q)UIT->')
    ACCEPT 120,IOPT
120   FORMAT(A1)
C
C    JUMP ON OPTION
C
    IF(IOPT.EQ.'C') CALL CALIB(NCHAN,NPTS,DRATE,NIDS)
    IF(IOPT.EQ.'Q') GOTO 2000
    IF(IOPT.EQ.'R') GOTO 500
    IF(IOPT.EQ.'T') CALL TAPE(NCHAN,NPTS,NIDS)
    GOTO 100

C
C ===== RUN TIME SETUP =====
C
C    TRAP "Q" FOR ABORT.
C
500   ISWFLG=0
    CALL SCCA(ISWFLG)
    CALL KEYON          !TURN OFF ECHO, CHECK FOR KEY PRESSES

C
C    INITIALIZE NBYTES FOR TAPE WRITE
C
    NBYTES=(NIDS+NCHAN*NPTS)*2

C
C    INITIALIZE DEVICES
C
    CALL LINOFF
    CALL CLINIT
    CALL DIINIT
    CALL DOINIT
    DO (ICLOCK=2,5)
    CLOCK=ICLOCK
    CALL PCINIT(CLOCK)
    FIN

C
C===== MAIN RUN TIME LOOP=====

```

```

C      DO (BLOCK=1,96)
C
C      ZERO ID ARRAY
C
C      CALL ZERO(ID,NIDS*2)
C
C      CHECK FOR "Q" AND "P"
C
C      CALL KEYS(KEYFLG)                !CHECK FOR KEY PRESSES
C      IF(KEYFLG.EQ.1)
C      CALL CLKILL
C      CALL MCERAS
C      CALL DIKILL
C      CALL DOKILL
C      CALL ADKILL
C      CALL KEYOFF
C      DO (ICLOCK=2,5)
C      CLOCK=ICLOCK
C      CALL PCKILL(CLOCK)
C      FIN
C      CALL EOF(0)
C      CALL EOF(0)
C      CALL SKIP(0,-2)
C      GOTO 100
C      FIN
C
C      IBLK=BLOCK
C      IF(((IBLK/16)*16).EQ.IBLK.AND.IBLK.NE.96)
C      TYPE *, ' Program paused '
C      CALL MCTEXT(20,200,'Pause — Press button to continue ')
C      CALL MCDISP(1)
C
C      WAIT FOR SUBJECT TO PRESS BUTTON
C
C      CALL DISET(0,'SINGLE',ISTATO)
C      CALL DISET(1,'SINGLE',ISTAT1)
C      CALL WAIT(ISTATO,ISTAT1)
C      CALL MCDISP(0)
C      CALL MCERAS
C      FIN
C
C      READ CONDITION, STORE IDS AND SELECT STIMULI
C
C      SUB    =COND(BLOCK,1)
C      SESS   =COND(BLOCK,2)
C      SOA    =COND(BLOCK,3)
C      DELAY  =COND(BLOCK,4)
C      CMVM   =COND(BLOCK,5)
C      MEMSIZ=COND(BLOCK,6)
C      STYPE  =COND(BLOCK,7)

```

```

C
ID(1)= SUB
ID(2)= SESS
ID(3)= BLOCK
ID(4)= DELAY
ID(5)= SOA
ID(6)= MEMSIZ
ID(7)= CMVM
ID(8)= YESBUT
ID(9)= NOBUT

C
C   INFORM EXPERIMENTER OF CONDITION
C
TYPE 30,BLOCK,DELAY,CMVM,MEMSIZ,SType
30  FORMAT($' Block ',I3,' Delay ',I2,' CMVM ',I1,' Memsiz ',I1,
+ ' PosNeg ',I1)

C
DO (K=1,10)

C
C   PERMUTE CM TARGETS
C
DO (LOOP =1,5)
J=ITWIXT(1,5)
DO (I=1,4)
TEMP1 = CMTARG(J,I)
CMTARG(J,I) = CMTARG(LOOP,I)
CMTARG(LOOP,I) = TEMP1
FIN
FIN

C
C   PERMUTE CM DISTRACTORS
C
DO (LOOP =1,5)
J=ITWIXT(1,5)
DO (I=1,4)
TEMP1 = CMDIST(J,I)
CMDIST(J,I) = CMDIST(LOOP,I)
CMDIST(LOOP,I) = TEMP1
FIN
FIN

C
C   PERMUTE VM STIMULI
C
DO (LOOP =1,10)
J=ITWIXT(1,10)
DO (I=1,4)
TEMP1 = VMSTIM(J,I)
VMSTIM(J,I) = VMSTIM(LOOP,I)
VMSTIM(LOOP,I) = TEMP1
FIN
FIN

```

```

      FIN
C
C   CHOOSE CM STIMULI
C
      IF(CMVM.EQ.1)
      DO (J=1, MEMSIZ)
      DO (I=1, 4)
      TARG(J,I)=CMTARG(J,I)
      FIN
      FIN
C
      DO(J=1, 5)
      DO(I=1, 4)
      DIST(J,I)=CMDIST(J,I)
      FIN
      FIN
C
      FIN
C
C   CHOOSE VM STIMULI
C
      IF(CMVM.EQ.2)
      DO (J=1, MEMSIZ)
      DO (I=1, 4)
      TARG(J,I)=VMSTIM(J,I)
      FIN
      FIN
C
      DO(J=MEMSIZ+1, MEMSIZ+5)
      DO(I=1, 4)
      DIST(J-MEMSIZ, I)=VMSTIM(J, I)
      FIN
      FIN
C
      FIN
C
C ===== PRESENT MEMORY SET =====
C
C
C   WRITE STIMULI TO MATROX AND STORE IDS
C
      DO(I=1, MEMSIZ)
C
      IF(MEMSIZ.EQ.2)
      XPOS=60+(I*35)
      YPOS=100
      FIN
C
      IF(MEMSIZ.EQ.4)
      XPOS=27+(I*35)

```

```

YPOS=100
FIN
C
DO (J=1,4)
WORD(J)=TARG(I,J)
FIN
C
DO(J=1,4)
ID(80+(4*I)+J)=TARG(I,J)
FIN
WORD(5)=0
CALL MCTEXT(XPOS,YPOS,WORD)
FIN
C
C
C
TURN ON STIMULUS
C
CALL MCSYNC          ! SUNCH DISPLAY
CALL MCDISP(1)        ! TURN ON STIMULUS
CALL PCSET(1+1,'SINGLE',3,3000,IFLAG1)
CALL WAIT(IFLAG1)     ! WAIT FOR 3 SECONDS
CALL MCDISP(0)        ! TURN OFF STIMULUS
CALL MCERAS          ! ERASE MATROX
C
C
C
WAIT FOR 1500 MSEC BEFORE PRESENTING PROBES
C
CALL PCSET(1+1,'SINGLE',3,1500,IFLAG1)
CALL WAIT(IFLAG1)
C
C ===== RUNNING MEMORY TRIALS =====
C
IF(DELAY.NE.0)
DO (ITRL=1,IABS(DELAY))
C
C
C
PREPARE STIMULI DURING PRE-STIMULUS INTERVAL
C
CALL PCSET(1+1,'SINGLE',3,5,IFLAG1)
POSNEG=ITWIXT(1,3)
IF(POSNEG.EQ.3) POSNEG=2
C
C
C
IF A POSITIVE TRIAL, REPEAT STIMULUS NBACK
IF A NEGATIVE TRIAL, SELECT A NEW STIMULUS
C
IF(POSNEG.EQ.1) ! POSITIVE TRIAL
IF(ITRL.LE.NBACK) JSTIM=ITWIXT(1,9)
IF(ITRL.GT.NBACK) JSTIM=LSTIM(NBACK)
FIN
C
IF(POSNEG.EQ.2) ! NEGATIVE TRIAL
10 JSTIM=ITWIXT(1,9)
IF(JSTIM.EQ.LSTIM(NBACK)) GOTO 10
FIN

```

```

C
C   STORE SUCCESSIVE STIMULI
C
LSTIM(3) = LSTIM(2)
LSTIM(2) = LSTIM(1)
LSTIM(1) = JSTIM
ID((ITRL*5)+5) = JSTIM
C
XPOS=120
YPOS=120
DIGIT(1)=NUM(JSTIM)
DIGIT(2)=0
IF (DELAY.GT.0) CALL MCTEXT(XPOS,YPOS,DIGIT)
C
CALL WAIT(IFLAG1)      ! WAIT OUT PRESTIMULUS INTERVAL
C
CALL MCSYNC             ! SYNCH DISPLAY
CALL MCDISP(1)          ! TURN ON STIMULUS
CALL PCSET(1+1,'SINGLE',3,1000,IFLAG1)
CALL DIRESP(RES,'SINGLE',2,1+1,RT,RSTAT)
CALL PCSET(1+2,'SINGLE',3,200,IFLAG2)
CALL WAIT(IFLAG2)
CALL MCDISP(0)          ! TURN OFF STIMULUS
CALL MCERAS             ! ERASE MATROX
CALL WAIT(IFLAG1)      !WAIT FOR END OF 1000 MSEC INTERVAL
C
CLASSIFY TRIAL AND STORE RT AND STIM TYPE (1=HIT, 2=CR, 3=FA, 4=MISS)
C
ID((ITRL*5)+6) = RT
ID((ITRL*5)+7) = POSNEG
ID((ITRL*5)+8) = 0
ID((ITRL*5)+9) = 0
C
IF(RSTAT.NE.0)
C
IF(RES.EQ.YESBUT)
ID((ITRL*5)+8) = 1
IF (ID((ITRL*5)+7).EQ.1) ID((ITRL*5)+9) = 1
IF (ID((ITRL*5)+7).EQ.2) ID((ITRL*5)+9) = 3
FIN
C
IF(RES.EQ.NOBT)
ID((ITRL*5)+8) = 2
IF (ID((ITRL*5)+7).EQ.2) ID((ITRL*5)+9) = 2
IF (ID((ITRL*5)+7).EQ.1) ID((ITRL*5)+9) = 4
FIN
C
FIN
C
FIN
FIN

```



```

C
C ===== STERNBERG PROBE TRIAL =====
C
C
C      START THE DIGITIZER
C
C      CALL ADSET(DATA,2,1,NCHAN,NPTS,3,DRATE,DSTAT)
C
C      WAIT OUT BASELINE CLOCK
C
C      CALL PCSET(1+1,'SINGLE',3,BASE,IFLAG1)
C      CALL WAIT(IFLAG1)
C
C      DELIVER RETRIEVAL CUE
C
C      IF(SOA.GE.200)
C      XPOS=120
C      YPOS=120
C      CALL MCCHAR(XPOS,YPOS,'*')
C      CALL MCSYNC          ! SYNCH DISPLAY
C      CALL DOSET(0)        ! TURN ON STIMULUS MARKER
C      CALL MCDISP(1)       ! TURN ON MATROX
C      CALL PCSET(1+1,'SINGLE',3,SOA,IFLAG1)
C      CALL PCSET(1+2,'SINGLE',3,200,IFLAG2)
C      CALL WAIT(IFLAG2)
C      CALL MCDISP(0)       ! TURN OFF MATROX
C      CALL DOCLR(0)        ! TURN OFF STIMULUS MARKER
C      CALL MCERAS         ! ERASE MATROX
C      CALL WAIT(IFLAG1)    ! WAIT FOR END OF SOA INTERVAL
C      FIN
C
C      PREPARE STIMULI DURING PRE-STIMULUS INTERVAL
C
C      CALL PCSET(1+1,'SINGLE',3,5,IFLAG1)
C      XPOS=112
C      YPOS=140
C
C      IF(STYPE.EQ.1)      ! POSITIVE TRIAL
C      I=ITWIXT(1,MEMSIZ)
C      DO(J=1,4)
C      WORD(J)=TARG(I,J)
C      ID(104+J)=TARG(I,J)
C      FIN
C      FIN
C
C      IF (STYPE.EQ.2)     ! NEGATIVE TRIAL
C      I=ITWIXT(1,5)
C      DO(J=1,4)
C      WORD(J)=DIST(I,J)
C      ID(104+J)=DIST(I,J)
C      FIN

```

```

FIN
WORD(5)=0
CALL MCTEXT(XPOS,YPOS,WORD)
CALL WAIT(IFLAG1)
C
CALL MCSYNC          ! SYNCH DISPLAY
CALL DOSET(0)        ! TURN ON STIMULUS MARKER
CALL MCDISP(1)       ! TURN ON DISPLAY
CALL PCSET(1+1,'SINGLE',3,3000,IFLAG1)
CALL DIRESP(RES,'SINGLE',2,1+1,RT,RSTAT)
CALL PCSET(1+2,'SINGLE',3,200,IFLAG2)
CALL WAIT(IFLAG2)
CALL MCDISP(0)       ! TURN OFF DISPLAY
CALL DOCLR(0)        ! TURN OFF STIMULUS MARKER
CALL MCERAS          ! CLEAR MARTOX
CALL WAIT(IFLAG1)    ! WAIT FOR END OF 3 SECOND INTERVAL
C
C CLASSIFY TRIAL AND STORE RT AND STIM TYPE (1=HIT, 2=CR, 3=FA, 4=MISS)
C
ID(101) = RT
ID(102) = STYPE
ID(103) = 0
ID(104) = 0
C
IF(RSTAT.NE.0)
IF(RES.EQ.YESBUT)
ID(103) = 1
IF (ID(102).EQ.1) ID(104) = 1
IF (ID(102).EQ.2) ID(104) = 3
FIN
C
IF(RES.EQ.NOBT)
ID(103) = 2
IF (ID(102).EQ.2) ID(104) = 2
IF (ID(102).EQ.1) ID(104) = 4
FIN
C
FIN
C
C CHECK IF ANY POINTS ARE OUT OF BOUNDS
C
ERROR=0
DO (IPTS=1,210)
DO (ICHAN=1,4)
IF(IABS(DATA(ICHAN,IPTS)).GE.2047) ERROR=ERROR+1
FIN
FIN
C
C WRITE DATA TO TAPE
C
CALL PUT(0,ID,NBYTES,MSTAT)

```

```

      TYPE 20, ID(101),ID(104)
20    FORMAT('  RT ',I4,' Corr ',I1)
      IF (ERROR.NE.0) TYPE *,'Error in last trial of digitized EEG'
      CALL DISP(NCHAN,NPTS)
      CALL WAIT(MSTAT)
C
      FIN
C
C    KILL THE DEVICES
C
      CALL CLKILL
      CALL MCERAS
      CALL DIKILL
      CALL DOKILL
      CALL ADKILL
      CALL KEYOFF
      DO (ICLOCK=2,5)
      CLOCK=ICLOCK
      CALL PCKILL(CLOCK)
      FIN
C
C    WRITE DOUBLE EOF AND SKIP BACK 1
C
      CALL EOF(0)
      CALL EOF(0)
      CALL SKIP(0,-2)
C
C    RETURN TO MAIN MENU
C
      GOTO 100
C
C ===== QUIT SECTION =====
C
C    TURN OFF THE DISPLAY AND EXIT PROGRAM
C
2000  CONTINUE
      CALL CLKILL
      CALL MCERAS
      CALL GQINIT
      CALL DIKILL
      CALL ADKILL
      CALL KEYOFF
      STOP
      END

```

```

C*****
C                                                    *
C                S U B R O U T I N E S                *
C                                                    *
C*****
CCC
CCC    SUBROUTINE CALIB
CCC
C
C    GATHERS FIVE CALIBRATION TRIALS
C
C    SUBROUTINE CALIB(NCHAN,NPTS,DRATE,NIDS)
C    IMPLICIT INTEGER (A-Z)
C
C    COMMON /DATA/ ID(108),DATA(5,210)
C    NBYTES=(NIDS+NCHAN*NPTS)*2
C
C    CALL LINOFF
C    CALL CLINIT
C    WRITE(7,10)
10    FORMAT(/////` TURN ON CALIBRATION PULSE`)
C    PAUSE `HIT <CR> TO CONTINUE`
C    DIGITIZE 5 TRIALS OF CALIBRATION DATA
C    DO (LOOP=1,5)
C    CALL ADSET(DATA,2,1,NCHAN,NPTS,3,DRATE,DSTAT)
C    CALL WAIT(DSTAT)          !WAIT FOR END OF DIGITIZING
C    CALL PUT(0,ID,NBYTES,MSTAT)
C    CALL WAIT(MSTAT)          !WAIT FOR END OF WRITING TO TAPE
C
C    DISPLAY DATA ON GT
C
C    CALL DISP(NCHAN,NPTS)
C
C    FIN
C    CALL EOF(0)
C    CALL EOF(0)
C    CALL SKIP(0,-2)
C    RETURN
C    END
C *****

```

```

CCC
CCC  DISP
CCC
      SUBROUTINE DISP(NCHAN,NPTS)
      COMMON /DATA/ ID(108),DATA(5,210)
C     SUBROUTINE TO DISPLAY WAVEFORM TO GT DISPLAY
C
C     DISPLAY DATA ON GT44
C
      CALL DEBASE(DATA,NCHAN,NPTS,2,1,NPTS,XBAR)
      CALL GQDISP(1,2,DATA,NCHAN,NPTS,-2)
      CALL DEBASE(DATA,NCHAN,NPTS,3,1,NPTS,XBAR)
      CALL GQDISP(2,3,DATA,NCHAN,NPTS,-2)
      CALL DEBASE(DATA,NCHAN,NPTS,4,1,NPTS,XBAR)
      CALL GQDISP(3,4,DATA,NCHAN,NPTS,-2)
      CALL DEBASE(DATA,NCHAN,NPTS,1,1,NPTS,XBAR)
      CALL GQDISP(4,1,DATA,NCHAN,NPTS,-2)
      CALL DEBASE(DATA,NCHAN,NPTS,5,1,NPTS,XBAR)
      CALL GQDISP(5,5,DATA,NCHAN,NPTS,-2)
      RETURN
      END
C =====

```

```

CCC
CCC ITWIXT
CCC
C
C RETURNS AN INTEGER IN THE RANGE GIVEN, RANDOMLY CHOSEN.
C
      INTEGER FUNCTION ITWIXT(MIN,MAX)
      IMPLICIT INTEGER (A-Z)
      REAL PAN
      COMMON /RAN/ S1,S2
100   ITWIXT=RAN(S1,S2)*(MAX-MIN+1)+MIN
      IF(ITWIXT.LT.MIN.OR.ITWIXT.GT.MAX) GOTO 100
      RETURN
      END
C =====

```

```

CCC
CCC  SUBROUTINE KEYS
CCC
C
C  CHECK FOR STOP OR QUIT COMMANDS (KEY SUBROUTINE)
C
C  SUBROUTINE KEYS(KEYFLG)
C
C      INTEGER KEYFLG
C      BYTE KEY
C      KEYFLG=0
C      KEY=KEYCHK(IDUM)
C      IF(KEY.NE.'P') GOTO 10
C      CALL KEYOFF
C      PAUSE '-- HIT <CR> TO CONTINUE --'
C      CALL KEYON
10  IF(KEY.NE.'Q') GOTO 20
C      CALL KEYOFF
C      WRITE(7,*) 'BLOCK ABORTED: BACK TO MENU (2 EOFs WRITTEN)'
C      KEYFLG=1
20  RETURN
C      END
C=====

```

```

CCC
CCC TAPE
CCC
C
C SUBROUTINE TO MANIPULATE TAPE.
C
      SUBROUTINE TAPE(NCHAN,NPTS,NIDS)
      IMPLICIT INTEGER (A-Z)
      BYTE IOPT1
      COMMON /DATA/ ID(108),DATA(5,210)

C
C FIND NBYTES FOR TAPE READ, SKIP A LINE
C
      NBYTES=(NIDS+NCHAN*NPTS)*2
      TYPE *,' '

C
C GET THE OPTION FROM THE USER
C
5      WRITE(7,10)
10     FORMAT('$TAPE: (E)XIT, (R)EWIND, ',
+         ' (L)ABEL, (S)KIP, (V)ERIFY, (W)EOF> ')
      READ(5,15) IOPT1
15     FORMAT(A1)
C
C JUMP ACCORDING TO OPTION
C
      IF(IOPT1.EQ.'E') RETURN
      IF(IOPT1.EQ.'R') CALL SKIP(0,0)
      IF(IOPT1.EQ.'W') CALL EOF(0)
      IF(IOPT1.EQ.'S') GOTO 50
      IF(IOPT1.EQ.'V') GOTO 100
      IF(IOPT1.EQ.'L') GOTO 300
      GOTO 5

C
C SKIP EOF OPTION
C
50     WRITE(7,55)
55     FORMAT('$NUMBER OF EOFS TO SKIP >')
      READ(5,*) IFILES
      CALL SKIP(0,IFILES)
      GOTO 5

C
C VERIFY TAPE OPTION - READ ONLY IDS
C
100    DO 200 NT=1,9000
      CALL GET(0,ID,NBYTES,MSTAT)
      CALL WAIT(MSTAT)
      IF(MSTAT.NE.1) GOTO 210
      IF(ID(1).NE."177777") GOTO 150
      WRITE(7,125) (ID(L),L=2,50)

```



```

125     FORMAT(' LABEL:',49A1)
        GOTO 200
150     WRITE(7,175) NT,(ID(K),K=1,108)
175     FORMAT(' RECORD: ',16/,11(10I6/))
C
C     DISPLAY DATA ON GT
C
        CALL DISP(NCHAN,NPTS)
C
200     CONTINUE
210     NT=NT-1
        GOTO 5
C
C WRITE A LABEL
C
300     CALL ZERO(ID,100)
        ID(1)="177777
        WRITE(7,311)
311     FORMAT('$LABEL (LESS THAN 50 CHARS) >')
        READ(5,310) (ID(L),L=2,50)
310     FORMAT(49A1)
        CALL PUT(0,ID,100,MSTAT)
        CALL WAIT(MSTAT)
        CALL ZERO(ID,100)
        GOTO 5
        END

```

## Appendix I

PMSM.FTX Source code for primary/secondary memory experiment used to collect ERPS. This is the FORTRAN source code (the output of the flecs preprocessor) and must be compiled using the FORTRAN compiler.

C  
 C PMSM.FTX  
 C  
 C MODIFICATION DATE: JAN 16, 1988  
 C  
 C PROGRAM TO CONTROL PRIMARY/SECONDARY MEMORY STUDY  
 C  
 C IDS OUTPUT BY THE PROGRAM:  
 C  
 C ID(1) - SUBJECT NUMBER  
 C ID(2) - SESSION  
 C ID(3) - BLOCK  
 C ID(4) - DELAY  
 C ID(5) - SOA OF RETRIEVAL CUE  
 C ID(6) - SIZE OF MEMORY SET  
 C ID(7) - CONSTANT (1) OR VARIED (2) MAPPING  
 C ID(8) - CODE FOR YES BUTTON  
 C ID(9) - CODE FOR NO BUTTON  
 C  
 C RUNNING MEMORY IDS  
 C  
 C TRIAL 1  
 C  
 C ID(10) - RUNNING MEMORY STIMULUS  
 C ID(11) - RT (FOR RUNNING MEMORY TRIAL)  
 C ID(12) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)  
 C ID(13) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)  
 C ID(14) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,  
 C 2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS  
 C  
 C TRIAL 2  
 C  
 C ID(15) - RUNNING MEMORY STIMULUS  
 C ID(16) - RT (FOR RUNNING MEMORY TRIAL)  
 C ID(17) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)  
 C ID(18) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)  
 C ID(19) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,  
 C 2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS  
 C  
 C TRIAL 3  
 C  
 C ID(20) - RUNNING MEMORY STIMULUS  
 C ID(21) - RT (FOR RUNNING MEMORY TRIAL)  
 C ID(22) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)  
 C ID(23) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)  
 C ID(24) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,  
 C 2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS  
 C  
 C TRIAL 4  
 C  
 C ID(25) - RUNNING MEMORY STIMULUS

C ID(26) - RT (FOR RUNNING MEMORY TRIAL)  
 C ID(27) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)  
 C ID(28) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)  
 C ID(29) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,  
 C 2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS  
 C  
 C TRIAL 5  
 C  
 C ID(30) - RUNNING MEMORY STIMULUS  
 C ID(31) - RT (FOR RUNNING MEMORY TRIAL)  
 C ID(32) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)  
 C ID(33) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)  
 C ID(34) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,  
 C 2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS  
 C  
 C TRIAL 6  
 C  
 C ID(35) - RUNNING MEMORY STIMULUS  
 C ID(36) - RT (FOR RUNNING MEMORY TRIAL)  
 C ID(37) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)  
 C ID(38) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)  
 C ID(39) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,  
 C 2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS  
 C  
 C TRIAL 7  
 C  
 C ID(40) - RUNNING MEMORY STIMULUS  
 C ID(41) - RT (FOR RUNNING MEMORY TRIAL)  
 C ID(42) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)  
 C ID(43) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)  
 C ID(44) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,  
 C 2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS  
 C  
 C TRIAL 8  
 C  
 C ID(45) - RUNNING MEMORY STIMULUS  
 C ID(46) - RT (FOR RUNNING MEMORY TRIAL)  
 C ID(47) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)  
 C ID(48) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)  
 C ID(49) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,  
 C 2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS  
 C  
 C TRIAL 9  
 C  
 C ID(50) - RUNNING MEMORY STIMULUS  
 C ID(51) - RT (FOR RUNNING MEMORY TRIAL)  
 C ID(52) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)  
 C ID(53) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)  
 C ID(54) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,  
 C 2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS  
 C

```

C TRIAL 10
C
C ID(55) - RUNNING MEMORY STIMULUS
C ID(56) - RT (FOR RUNNING MEMORY TRIAL)
C ID(57) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)
C ID(58) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)
C ID(59) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,
C          2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS
C
C TRIAL 11
C
C ID(60) - RUNNING MEMORY STIMULUS
C ID(61) - RT (FOR RUNNING MEMORY TRIAL)
C ID(62) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)
C ID(63) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)
C ID(64) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,
C          2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS
C
C TRIAL 12
C
C ID(65) - RUNNING MEMORY STIMULUS
C ID(66) - RT (FOR RUNNING MEMORY TRIAL)
C ID(67) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)
C ID(68) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)
C ID(69) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,
C          2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS
C
C TRIAL 13
C
C ID(70) - RUNNING MEMORY STIMULUS
C ID(71) - RT (FOR RUNNING MEMORY TRIAL)
C ID(72) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)
C ID(73) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)
C ID(74) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,
C          2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS
C
C TRIAL 14
C
C ID(75) - RUNNING MEMORY STIMULUS
C ID(76) - RT (FOR RUNNING MEMORY TRIAL)
C ID(77) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)
C ID(78) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)
C ID(79) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,
C          2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS
C
C TRIAL 15
C
C ID(80) - RUNNING MEMORY STIMULUS
C ID(81) - RT (FOR RUNNING MEMORY TRIAL)
C ID(82) - STIM TYPE FOR RUNNING MEMORY (1=TARGET, 2=NONTARGET)
C ID(83) - RESPONSE FOR RUNNING MEMORY (0=NONE, 1=TARGET, 2=NONTARGET)

```

```

C   ID(84) - CORRECTNESS FOR RUNNING MEMORY: 0=NO RESPONSE, 1=HIT,
C           2=CORRECT REJECTION, 3=FALSE ALARM, 4=MISS
C
C   STERNBERG IDS
C
C   ID(85) - MEMORY SET WORD #1, LETTER #1
C   ID(86) - MEMORY SET WORD #1, LETTER #2
C   ID(87) - MEMORY SET WORD #1, LETTER #3
C   ID(88) - MEMORY SET WORD #1, LETTER #4
C
C   ID(89) - MEMORY SET WORD #2, LETTER #1
C   ID(90) - MEMORY SET WORD #2, LETTER #2
C   ID(91) - MEMORY SET WORD #2, LETTER #3
C   ID(92) - MEMORY SET WORD #2, LETTER #4
C
C   ID(93) - MEMORY SET WORD #3, LETTER #1
C   ID(94) - MEMORY SET WORD #3, LETTER #2
C   ID(95) - MEMORY SET WORD #3, LETTER #3
C   ID(96) - MEMORY SET WORD #3, LETTER #4
C
C   ID(97) - MEMORY SET WORD #4, LETTER #1
C   ID(98) - MEMORY SET WORD #4, LETTER #2
C   ID(99) - MEMORY SET WORD #4, LETTER #3
C   ID(100) - MEMORY SET WORD #4, LETTER #4
C
C   ID(101) - RT (FOR STERNBERG TRIAL)
C   ID(102) - STIM TYPE (1=TARGET, 2=NONTARGET)
C   ID(103) - RESPONSE (0=NONE, 1=TARGET, 2=NONTARGET)
C   ID(104) - CORRECTNESS: 0=NO RESPONSE, 1=HIT, 2=CORRECT REJECTION,
C           3=FALSE ALARM, 4=MISS
C
C   ID(105) - STIMULUS PRESENTED, LETTER #1
C   ID(106) - STIMULUS PRESENTED, LETTER #2
C   ID(107) - STIMULUS PRESENTED, LETTER #3
C   ID(108) - STIMULUS PRESENTED, LETTER #4
C
C-----
C
C
C   IMPLICIT INTEGER (A-Z)
C   INTEGER COND(96,7),LSTIM(3),ISEED(2)
C   BYTE NAME(17),ERR,TEMP1,WORD(5),NUM(9),DIGIT(2),YES,NO,
C +CMTARG(5,4),CMDIST(5,4),VMSTIM(10,4),TARG(5,4),DIST(5,4)
C   REAL SEED,RAN
C   EQUIVALENCE (SEED,ISEED),(ISEED(1),S1),(ISEED(2),S2)
C   COMMON /RAN/ S1,S2
C   COMMON /DATA/ ID(108),DATA(5,210)
C
C   DATA STMDUR,NBACK/200,2/
C   DATA NCHAN,NPTS,DRATE,NIDS,BASE/5,210,10,108,100/
C   DATA NUM/'1','2','3','4','5','6','7','8','9'/

```

```

C
C      GET A RANDOM SEED AND WARM UP RANDOM NUMBER GENERATOR
C
      SEED=SECNDS(0)
      DO 32765 RANDOM=1,2000
      CALL RAN(S1,S2)
32765 CONTINUE
C
C      INITIALIZE MATROX
C
32766 CALL MCINIT(1)
C
C      SET UP DISPLAY ASSIGNMENTS
C
      CALL GQINIT
      CALL GQTTL(' PM/SM EXPERIMENT')
      CALL GQLABL(1,'FZ ELECTRODE')
      CALL GQLABL(2,'CZ ELECTRODE')
      CALL GQLABL(3,'PZ ELECTRODE')
      CALL GQLABL(4,'EOG ELECTRODE')
      CALL GQLABL(5,'STM MARKER')
C
C      ASSIGN STIMULUS FILE FROM DISC
C
      TYPE *,'INPUT STIMULUS FILE FOR THIS SUBJECT'
      CALL GETSTR(5,NAME,16,ERR)
      CALL ASSIGN(40,NAME,0,'OLD')
C
C      READ STIMULUS FILE
C
      DO 32763 I=1,5
      READ(40,1) (CMTARG(I,J),J=1,4)
32763 CONTINUE
C
32764 DO 32761 I=1,5
      READ(40,1) (CMDIST(I,J),J=1,4)
32761 CONTINUE
C
32762 DO 32759 I=1,10
      READ(40,1) (VMSTIM(I,J),J=1,4)
32759 CONTINUE
C
32760 READ(40,2) YES,NO
      IF(.NOT.(YES.EQ.'/')) GO TO 32758
      YESBUT=1
      NOBUT=0
32758 IF(.NOT.(YES.EQ.'Z')) GO TO 32757
      YESBUT=0
      NOBUT=1
C
32757 DO 32755 J=1,96

```

```

      READ(40,3) (COND(J,I),I=1,7)
32755 CONTINUE
C
32756 CONTINUE
1   FORMAT(4A1)
2   FORMAT(2A1)
3   FORMAT(7I5)
C
C ----- GET THE OPTION FROM THE USER -----
C
C   GET THE OPTION
C
100  CONTINUE
     TYPE 110
110  FORMAT(// '$ MENU: (R)UN, (T)APE, (C)ALIB, (Q)UIT->')
     ACCEPT 120,IOPT
120  FORMAT(A1)
C
C   JUMP ON OPTION
C
     IF(IOPT.EQ.'C') CALL CALIB(NCHAN,NPTS,DRA TE,NIDS)
     IF(IOPT.EQ.'Q') GOTO 2000
     IF(IOPT.EQ.'R') GOTO 500
     IF(IOPT.EQ.'T') CALL TAPE(NCHAN,NPTS,NIDS)
     GOTO 100
C
C ----- RUN TIME SETUP -----
C
C   TRAP "Q" FOR ABORT.
C
500  ISWFLG=0
     CALL SCCA(ISWFLG)
     CALL KEYON           !TURN OFF ECHO, CHECK FOR KEY PRESSES
C
C   INITIALIZE NBYTES FOR TAPE WRITE
C
     NBYTES=(NIDS+NCHAN*NPTS)*2
C
C   INITIALIZE DEVICES
C
     CALL LINOFF
     CALL CLINIT
     CALL DIINIT
     CALL DOINIT
     DO 32753 ICLOCK=2,5
     CLOCK=ICLOCK
     CALL PCINIT(CLOCK)
32753 CONTINUE
C
C----- MAIN RUN TIME LOOP-----
C

```



```

32754 DO 32751 BLOCK=1,96
C
C     ZERO ID ARRAY
C
C     CALL ZERO(ID,NIDS*2)
C
C     CHECK FOR "Q" AND "P"
C
C     CALL KEYS(KEYFLG)                                !CHECK FOR KEY PRESSES
C     IF(.NOT.(KEYFLG.EQ.1)) GO TO 32750
C     CALL CLKILL
C     CALL MCERAS
C     CALL DIKILL
C     CALL DOKILL
C     CALL ADKILL
C     CALL KEYOFF
C     DO 32748 ICLOCK=2,5
C     CLOCK=ICLOCK
C     CALL PCKILL(CLOCK)
32748 CONTINUE
32749 CALL EOF(0)
C     CALL EOF(0)
C     CALL SKIP(0,-2)
C     GOTO 100
C
32750 IBLK=BLOCK
C     IF(.NOT.(((IBLK/16)*16).EQ.IBLK.AND.IBLK.NE.96)) GO TO 32747
C     TYPE *, ' Program paused '
C     CALL MCTEXT(20,200,'Pause -- Press button to continue ')
C     CALL MCDISP(1)
C
C     WAIT FOR SUBJECT TO PRESS BUTTON
C
C     CALL DISET(0,'SINGLE',ISTAT0)
C     CALL DISET(1,'SINGLE',ISTAT1)
C     CALL WAIT(ISTAT0,ISTAT1)
C     CALL MCDISP(0)
C     CALL MCERAS
C
C     READ CONDITION, STORE IDS AND SELECT STIMULI
C
32747 SUB    =COND(BLOCK,1)
C     SESS    =COND(BLOCK,2)
C     SOA     =COND(BLOCK,3)
C     DELAY   =COND(BLOCK,4)
C     CMVM    =COND(BLOCK,5)
C     MEMSIZ  =COND(BLOCK,6)
C     STYPE   =COND(BLOCK,7)
C
C     ID(1)= SUB
C     ID(2)= SESS

```

```

ID(3)= BLOCK
ID(4)= DELAY
ID(5)= SOA
ID(6)= MEMSIZ
ID(7)= CMVM
ID(8)= YESBUT
ID(9)= NOBUT
C
C   INFORM EXPERIMENTER OF CONDITION
C
TYPE 30,BLOCK,DELAY,CMVM,MEMSIZ,STYPE
30  FORMAT($~ Block ~,I3,~ Delay ~,I2,~ CMVM ~,I1,~ Memsiz ~,I1,
+~ PosNeg ~,I1)
C
DO 32745 K=1,10
C
C   PERMUTE CM TARGETS
C
DO 32743 LOOP =1,5
J=ITWIXT(1,5)
DO 32741 I=1,4
TEMP1 = CMTARG(J,I)
CMTARG(J,I) = CMTARG(LOOP,I)
CMTARG(LOOP,I) = TEMP1
32741 CONTINUE
32742 CONTINUE
32743 CONTINUE
C
C   PERMUTE CM DISTRACTORS
C
32744 DO 32739 LOOP =1,5
J=ITWIXT(1,5)
DO 32737 I=1,4
TEMP1 = CMDIST(J,I)
CMDIST(J,I) = CMDIST(LOOP,I)
CMDIST(LOOP,I) = TEMP1
32737 CONTINUE
32738 CONTINUE
32739 CONTINUE
C
C   PERMUTE VM STIMULI
C
32740 DO 32735 LOOP =1,10
J=ITWIXT(1,10)
DO 32733 I=1,4
TEMP1 = VMSTIM(J,I)
VMSTIM(J,I) = VMSTIM(LOOP,I)
VMSTIM(LOOP,I) = TEMP1
32733 CONTINUE
32734 CONTINUE
32735 CONTINUE

```

```

32736 CONTINUE
32745 CONTINUE
C
C    CHOOSE CM STIMULI
C
32746 IF(.NOT.(CMVM.EQ.1)) GO TO 32732
      DO 32730 J=1, MEMSIZ
      DO 32728 I=1, 4
      TARG(J,I)=CMTARG(J,I)
32728 CONTINUE
32729 CONTINUE
32730 CONTINUE
C
32731 DO 32726 J=1, 5
      DO 32724 I=1, 4
      DIST(J,I)=CMDIST(J,I)
32724 CONTINUE
32725 CONTINUE
32726 CONTINUE
C
32727 CONTINUE
C
C    CHOOSE VM STIMULI
C
32732 IF(.NOT.(CMVM.EQ.2)) GO TO 32723
      DO 32721 J=1, MEMSIZ
      DO 32719 I=1, 4
      TARG(J,I)=VMSTIM(J,I)
32719 CONTINUE
32720 CONTINUE
32721 CONTINUE
C
32722 DO 32717 J=MEMSIZ+1, MEMSIZ+5
      DO 32715 I=1, 4
      DIST(J-MEMSIZ,I)=VMSTIM(J,I)
32715 CONTINUE
32716 CONTINUE
32717 CONTINUE
C
32718 CONTINUE
C
C ===== PRESENT MEMORY SET =====
C
C
C    WRITE STIMULI TO MATROX AND STORE IDS
C
32723 DO 32713 I=1, MEMSIZ
C
      IF(.NOT.(MEMSIZ.EQ.2)) GO TO 32712
      XPOS=60+(I*35)
      YPOS=100

```

```

C
32712 IF(.NOT.(MEMSIZ.EQ.4)) GO TO 32711
      XPOS=27+(I*35)
      YPOS=100
C
32711 DO 32709 J=1,4
      WORD(J)=TARG(I,J)
32709 CONTINUE
C
32710 DO 32707 J=1,4
      ID(80+(4*I)+J)=TARG(I,J)
32707 CONTINUE
32708 WORD(5)=0
      CALL MCTEXT(XPOS,YPOS,WORD)
32713 CONTINUE
C
C      TURN ON STIMULUS
C
32714 CALL MCSYNC          ! SUNCH DISPLAY
      CALL MCDISP(1)       ! TURN ON STIMULUS
      CALL PCSET(1+1,'SINGLE',3,3000,IFLAG1)
      CALL WAIT(IFLAG1)    ! WAIT FOR 3 SECONDS
      CALL MCDISP(0)       ! TURN OFF STIMULUS
      CALL MCERAS          ! ERASE MATROX
C
C      WAIT FOR 1500 MSEC BEFORE PRESENTING PROBES
C
      CALL PCSET(1+1,'SINGLE',3,1500,IFLAG1)
      CALL WAIT(IFLAG1)
C
C ----- RUNNING MEMORY TRIALS -----
C
      IF(.NOT.(DELAY.NE.0)) GO TO 32706
      DO 32704 ITRL=1,IABS(DELAY)
C
C      PREPARE STIMULI DURING PRE-STIMULUS INTERVAL
C
      CALL PCSET(1+1,'SINGLE',3,5,IFLAG1)
      POSNEG=ITWIXT(1,3)
      IF(POSNEG.EQ.3) POSNEG=2
C
C      IF A POSITIVE TRIAL, REPEAT STIMULUS NBACK
C      IF A NEGATIVE TRIAL, SELECT A NEW STIMULUS
C
      IF(.NOT.(POSNEG.EQ.1)) GO TO 32703
      IF(ITRL.LE.NBACK) JSTIM=ITWIXT(1,9)
      IF(ITRL.GT.NBACK) JSTIM=LSTIM(NBACK)
C
32703 IF(.NOT.(POSNEG.EQ.2)) GO TO 32702
10    JSTIM=ITWIXT(1,9)
      IF(JSTIM.EQ.LSTIM(NBACK)) GOTO 10

```

```

C
C   STORE SUCCESSIVE STIMULI
C
32702 LSTIM(3) = LSTIM(2)
      LSTIM(2) = LSTIM(1)
      LSTIM(1) = JSTIM
      ID((ITRL*5)+5) = JSTIM
C
      XPOS=120
      YPOS=120
      DIGIT(1)=NUM(JSTIM)
      DIGIT(2)=0
      IF (DELAY.GT.0) CALL MCTEXT(XPOS,YPOS,DIGIT)
C
      CALL WAIT(IFLAG1)      ! WAIT OUT PRESTIMULUS INTERVAL
C
      CALL MCSYNC            ! SYNCH DISPLAY
      CALL MCDISP(1)         ! TURN ON STIMULUS
      CALL PCSET(1+1,'SINGLE',3,1000,IFLAG1)
      CALL DIRESP(RESPI,'SINGLE',2,1+1,RT,RSTAT)
      CALL PCSET(1+2,'SINGLE',3,200,IFLAG2)
      CALL WAIT(IFLAG2)
      CALL MCDISP(0)         ! TURN OFF STIMULUS
      CALL MCERAS            ! ERASE MATROX
      CALL WAIT(IFLAG1)      !WAIT FOR END OF 1000 MSEC INTERVAL
C
C   CLASSIFY TRIAL AND STORE RT AND STIM TYPE (1=HIT, 2=CR, 3=FA, 4=MISS)
C
      ID((ITRL*5)+6) = RT
      ID((ITRL*5)+7) = POSNEG
      ID((ITRL*5)+8) = 0
      ID((ITRL*5)+9) = 0
C
      IF(.NOT.(RSTAT.NE.0)) GO TO 32701
C
      IF(.NOT.(RESP.EQ.YESBUT)) GO TO 32700
      ID((ITRL*5)+8) = 1
      IF (ID((ITRL*5)+7).EQ.1) ID((ITRL*5)+9) = 1
      IF (ID((ITRL*5)+7).EQ.2) ID((ITRL*5)+9) = 3
C
32700 IF(.NOT.(RESP.EQ.NOBT)) GO TO 32699
      ID((ITRL*5)+8) = 2
      IF (ID((ITRL*5)+7).EQ.2) ID((ITRL*5)+9) = 2
      IF (ID((ITRL*5)+7).EQ.1) ID((ITRL*5)+9) = 4
C
32699 CONTINUE
C
32701 CONTINUE
32704 CONTINUE
32705 CONTINUE
C

```

```

C ===== STERNBERG PROBE TRIAL =====
C
C
C   START THE DIGITIZER
C
32706 CALL ADSET(DATA,2,1,NCHAN,NPTS,3,D RATE,DSTAT)
C
C   WAIT OUT BASELINE CLOCK
C
C   CALL PCSET(1+1,'SINGLE',3,BASE,IFLAG1)
C   CALL WAIT(IFLAG1)
C
C   DELIVER RETRIEVAL CUE
C
C   IF(.NOT.(SOA.GE.200)) GO TO 32698
C   XPOS=120
C   YPOS=120
C   CALL MCCHAR(XPOS,YPOS,'*')
C   CALL MCSYNC          ! SYNCH DISPLAY
C   CALL DOSET(0)         ! TURN ON STIMULUS MARKER
C   CALL MCDISP(1)        ! TURN ON MATROX
C   CALL PCSET(1+1,'SINGLE',3,SOA,IFLAG1)
C   CALL PCSET(1+2,'SINGLE',3,200,IFLAG2)
C   CALL WAIT(IFLAG2)
C   CALL MCDISP(0)        ! TURN OFF MATROX
C   CALL DOCLR(0)         ! TURN OFF STIMULUS MARKER
C   CALL MCERAS          ! ERASE MATROX
C   CALL WAIT(IFLAG1)     ! WAIT FOR END OF SOA INTERVAL
C
C   PREPARE STIMULI DURING PRE-STIMULUS INTERVAL
C
32698 CALL PCSET(1+1,'SINGLE',3,5,IFLAG1)
C   XPOS=112
C   YPOS=140
C
C   IF(.NOT.(STYPE.EQ.1)) GO TO 32697
C   I=ITWIXT(1,MEMSIZ)
C   DO 32695 J=1,4
C   WORD(J)=TARG(I,J)
C   ID(104+J)=TARG(I,J)
32695 CONTINUE
32696 CONTINUE
C
32697 IF(.NOT.(STYPE.EQ.2)) GO TO 32694
C   I=ITWIXT(1,5)
C   DO 32692 J=1,4
C   WORD(J)=DIST(I,J)
C   ID(104+J)=DIST(I,J)
32692 CONTINUE
32693 CONTINUE
32694 WORD(5)=0

```

```

CALL MCTEXT(XPOS,YPOS,WORD)
CALL WAIT(IFLAG1)
C
CALL MCSYNC          ! SYNCH DISPLAY
CALL DOSET(0)        ! TURN ON STIMULUS MARKER
CALL MCDISP(1)       ! TURN ON DISPLAY
CALL PCSET(1+1,'SINGLE',3,3000,IFLAG1)
CALL DIRESP(RESPI,'SINGLE',2,1+1,RT,RSTAT)
CALL PCSET(1+2,'SINGLE',3,200,IFLAG2)
CALL WAIT(IFLAG2)
CALL MCDISP(0)       ! TURN OFF DISPLAY
CALL DOCLR(0)        ! TURN OFF STIMULUS MARKER
CALL MCERAS         ! CLEAR MARTOX
CALL WAIT(IFLAG1)    ! WAIT FOR END OF 3 SECOND INTERVAL
C
C CLASSIFY TRIAL AND STORE RT AND STIM TYPE (1=HIT, 2=CR, 3=FA, 4=MISS)
C
ID(101) = RT
ID(102) = STYPE
ID(103) = 0
ID(104) = 0
C
IF(.NOT.(RSTAT.NE.0)) GO TO 32691
IF(.NOT.(RESP.EQ.YESBUT)) GO TO 32690
ID(103) = 1
IF (ID(102).EQ.1) ID(104) = 1
IF (ID(102).EQ.2) ID(104) = 3
C
32690 IF(.NOT.(RESP.EQ.NOBT)) GO TO 32689
ID(103) = 2
IF (ID(102).EQ.2) ID(104) = 2
IF (ID(102).EQ.1) ID(104) = 4
C
32689 CONTINUE
C
C CHECK IF ANY POINTS ARE OUT OF BOUNDS
C
32691 ERROR=0
DO 32687 IPTS=1,210
DO 32685 ICHAN=1,4
IF(IABS(DATA(ICHAN,IPTS)).GE.2047) ERROR=ERROR+1
32685 CONTINUE
32686 CONTINUE
32687 CONTINUE
C
C WRITE DATA TO TAPE
C
32688 CALL PUT(0,ID,NBYTES,MSTAT)
TYPE 20, ID(101),ID(104)
20 FORMAT(' RT ',I4,' Corr ',I1)
IF (ERROR.NE.0) TYPE *,'Error in last trial of digitized EEG'

```

```

        CALL DISP(NCHAN,NPTS)
        CALL WAIT(MSTAT)
C
32751 CONTINUE
C
C      KILL THE DEVICES
C
32752 CALL CLKILL
      CALL MCERAS
      CALL DIKILL
      CALL DOKILL
      CALL ADKILL
      CALL KEYOFF
      DO 32683 ICLOCK=2,5
      CLOCK=ICLOCK
      CALL PCKILL(CLOCK)
32683 CONTINUE
C
C      WRITE DOUBLE EOF AND SKIP BACK 1
C
32684 CALL EOF(0)
      CALL EOF(0)
      CALL SKIP(0,-2)
C
C      RETURN TO MAIN MENU
C
      GOTO 100
C
C ===== QUIT SECTION =====
C
C      TURN OFF THE DISPLAY AND EXIT PROGRAM
C
2000  CONTINUE
      CALL CLKILL
      CALL MCERAS
      CALL GQINIT
      CALL DIKILL
      CALL ADKILL
      CALL KEYOFF
      STOP
      END

```



```

C*****
C                                                    *
C              S U B R O U T I N E S                *
C                                                    *
C*****
CCC
CCC  SUBROUTINE CALIB
CCC
C
C  GATHERS FIVE CALIBRATION TRIALS
C
C  SUBROUTINE CALIB(NCHAN,NPTS,DRATE,NIDS)
C  IMPLICIT INTEGER (A-Z)
C
C  COMMON /DATA/ ID(108),DATA(5,210)
C  NBYTES=(NIDS+NCHAN*NPTS)*2
C
C  CALL LINOFF
C  CALL CLINIT
C  WRITE(7,10)
10  FORMAT(/////` TURN ON CALIBRATION PULSE`)
C  PAUSE `HIT <CR> TO CONTINUE`
C  DIGITIZE 5 TRIALS OF CALIBRATION DATA
C  DO 32765 LOOP=1,5
C  CALL ADSET(DATA,2,1,NCHAN,NPTS,3,DRATE,DSTAT)
C  CALL WAIT(DSTAT)          !WAIT FOR END OF DIGITIZING
C  CALL PUT(0,ID,NBYTES,MSTAT)
C  CALL WAIT(MSTAT)          !WAIT FOR END OF WRITTING TO TAPE
C
C  DISPLAY DATA ON GT
C
C  CALL DISP(NCHAN,NPTS)
C
32765 CONTINUE
32766 CALL EOF(0)
C  CALL EOF(0)
C  CALL SKIP(0,-2)
C  RETURN
C  END
C =====

```

```

CCC
CCC  DISP
CCC
      SUBROUTINE DISP(NCHAN,NPTS)
      COMMON /DATA/ ID(108),DATA(5,210)
C      SUBROUTINE TO DISPLAY WAVEFORM TO GT DISPLAY
C
C      DISPLAY DATA ON GT44
C
      CALL DEBASE(DATA,NCHAN,NPTS,2,1,NPTS,XBAR)
      CALL GQDISP(1,2,DATA,NCHAN,NPTS,-2)
      CALL DEBASE(DATA,NCHAN,NPTS,3,1,NPTS,XBAR)
      CALL GQDISP(2,3,DATA,NCHAN,NPTS,-2)
      CALL DEBASE(DATA,NCHAN,NPTS,4,1,NPTS,XBAR)
      CALL GQDISP(3,4,DATA,NCHAN,NPTS,-2)
      CALL DEBASE(DATA,NCHAN,NPTS,1,1,NPTS,XBAR)
      CALL GQDISP(4,1,DATA,NCHAN,NPTS,-2)
      CALL DEBASE(DATA,NCHAN,NPTS,5,1,NPTS,XBAR)
      CALL GQDISP(5,5,DATA,NCHAN,NPTS,-2)
      RETURN
      END
C =====

```

```

CCC
CCC ITWIXT
CCC
C
C RETURNS AN INTEGER IN THE RANGE GIVEN, RANDOMLY CHOSEN.
C
      INTEGER FUNCTION ITWIXT(MIN,MAX)
      IMPLICIT INTEGER (A-Z)
      REAL RAN
      COMMON /RAN/ S1,S2
100    ITWIXT=RAN(S1,S2)*(MAX-MIN+1)+MIN
      IF(ITWIXT.LT.MIN.OR.ITWIXT.GT.MAX) GOTO 100
      RETURN
      END
C =====

```

```

CCC
CCC  SUBROUTINE KEYS
CCC
C
C    CHECK FOR STOP OR QUIT COMMANDS (KEY SUBROUTINE)
C
C    SUBROUTINE KEYS(KEYFLG)
C
C        INTEGER KEYFLG
C        BYTE KEY
C        KEYFLG=0
C        KEY=KEYCHK(IDUM)
C        IF(KEY.NE.'P') GOTO 10
C        CALL KEYOFF
C        PAUSE '— HIT <CR> TO CONTINUE --'
C        CALL KEYON
10    IF(KEY.NE.'Q') GOTO 20
C        CALL KEYOFF
C        WRITE(7,*) 'BLOCK ABORTED: BACK TO MENU (2 EOFs WRITTEN)'
C        KEYFLG=1
20    RETURN
C    END
C=====

```

```

CCC
CCC TAPE
CCC
C
C SUBROUTINE TO MANIPULATE TAPE.
C
      SUBROUTINE TAPE(NCHAN,NPTS,NIDS)
      IMPLICIT INTEGER (A-Z)
      BYTE IOPT1
      COMMON /DATA/ ID(108),DATA(5,210)
C
C FIND NBYTES FOR TAPE READ, SKIP A LINE
C
      NBYTES=(NIDS+NCHAN*NPTS)*2
      TYPE *,' '
C
C GET THE OPTION FROM THE USER
C
5      WRITE(7,10)
10     FORMAT('TAPE: (E)XIT, (R)EWIND, ',
+          ' (L)ABEL, (S)KIP, (V)ERIFY, (W)EOF> ')
      READ(5,15) IOPT1
15     FORMAT(A1)
C
C JUMP ACCORDING TO OPTION
C
      IF(IOPT1.EQ.'E') RETURN
      IF(IOPT1.EQ.'R') CALL SKIP(0,0)
      IF(IOPT1.EQ.'W') CALL EOF(0)
      IF(IOPT1.EQ.'S') GOTO 50
      IF(IOPT1.EQ.'V') GOTO 100
      IF(IOPT1.EQ.'L') GOTO 300
      GOTO 5
C
C SKIP EOF OPTION
C
50     WRITE(7,55)
55     FORMAT('NUMBER OF EOFs TO SKIP >')
      READ(5,*) IFILES
      CALL SKIP(0,IFILES)
      GOTO 5
C
C VERIFY TAPE OPTION - READ ONLY IDS
C
100    DO 200 NT=1,9000
      CALL GET(0,ID,NBYTES,MSTAT)
      CALL WAIT(MSTAT)
      IF(MSTAT.NE.1) GOTO 210
      IF(ID(1).NE."177777") GOTO 150
      WRITE(7,125) (ID(L),L=2,50)
125    FORMAT(' LABEL:',49A1)

```

```

        GOTO 200
150     WRITE(7,175) NT,(ID(K),K=1,108)
175     FORMAT(' RECORD: ',I6,/,11(10I6/))
C
C     DISPLAY DATA ON GT
C
        CALL DISP(NCHAN,NPTS)
C
200     CONTINUE
210     NT=NT-1
        GOTO 5
C
C WRITE A LABEL
C
300     CALL ZERO(ID,100)
        ID(1)="177777"
        WRITE(7,311)
311     FORMAT('$LABEL (LESS THAN 50 CHARS) >')
        READ(5,310) (ID(L),L=2,50)
310     FORMAT(49A1)
        CALL PUT(0,ID,100,MSTAT)
        CALL WAIT(MSTAT)
        CALL ZERO(ID,100)
        GOTO 5
        END

```

## Appendix J

List of Subroutine calls not included in the subroutine section of PMSM.FLX.  
A description of the subroutine function and the arguments is included.

Adkill:            - Abort an A/D Conversion in Progress

Adkill -- Abort an A/D Conversion in Progress

Format:

CALL adkill

Description:

The ADKILL Subroutine aborts any ADSET that might be in progress at the time. It is useful only in those cases where the program needs to abort any I/O activity that might be in progress before Exiting. When it is known that no ADSET is in progress, it is not necessary to Call this Subroutine before Exiting. The "Stat" variable for the ADSET Call Aborted is not Set to One by this Call.



AdSet                - Digitize a Block of Data

AdSet -- Digitize a Block of Data

Format:

CALL adset  
    (buffer,gain,start,nchan,npts,rate,count,stat)

Where:

buffer    (Integer Array, Nchan by Npts)  
          The Buffer into which the the Digitized Data are  
          stored.

gain      (Int Val, 0-3) Programmable Gain Select  
          0:    Multiply Input by 1    (-10 to 10 Volt Range)  
          1:    Multiply Input by 2    (-5 to 5 Volt Range)  
          2:    Multiply Input by 4    (-2.5 to 2.5 Volt Range)  
          3:    Multiply Input by 8    (-1.25 to 1.25 Volt Range)

start     (Int Val, 1-16) Channel to Start each Scan With

nchan     (Int Val, 1-16) Number of Channels to Sample  
          on each Scan

npts      (Int Val) The Number of Scans to do, or, the number  
          of Points in each Sweep. (Npts x Nchan = Buffer Size)

rate      (Int Val, 0-7) Basic Tick Rate for the Inter Scan Clock.  
          Rates Are:  
          0:    1    Microsecond            1    Megahertz  
          1:    10   Microseconds          100 Kiloherzt  
          2:    100 Microseconds          10   Kiloherzt  
          3:    1    Millisecond            1    Kiloherzt  
          4:    10   Milliseconds          100 Hertz  
          5:    100 Milliseconds          10   Hertz  
          6:    1    Second                1    Hertz  
          7:    10   Seconds               100 Millihertz

count     (Int Val) Number of Units of "Rate" to count off  
          between each scan.

stat      (Int Var) This Variable keeps the user informed of the  
          status of the A/D Conversion. When the ADSET is  
          executed, this variable is cleared, (that is, assigned  
          the value Zero). When the last Scan of the Sweep is  
          finished and the user's data array is full, "Stat" is  
          assigned the Value One...

          0:    A/D Conversion in Progress  
          1:    A/D Converter Sweep is Finished

Description:

The ADSET Subroutine permits its user to Digitize a Block of Data composed of "Npts" sets of "Nchan" Samples. The Call works as Follows. When an ADSET is Executed, "Nchan" Channels starting at Channel "Nchan" are Digitized right away.  
A Set of Samples across several channels digitized at (virtually) the

same time such as this will be referred to herein as a "Scan". Then an interval determined by "Rate" and "Count" is clocked off (by the A/D System's built in clock), and another Scan is performed. This process continues until "Npts" Scans have been performed. A complete set of clock scheduled Scans as described above is referred to as a "Sweep".

The Array into which the Data will go should be DIMENSIONED to contain "Nchan" by "Npts" Elements. BUFFER(I,J) will address the datum read from Channel "I" during the "J"th Scan.

The Values returned for each digitized value are 12 Bit Signed Integers. A value of -10 Volts presented to the A/D Inputs will result in a value of -2048 being returned by Adset when that channel is sampled and digitized. A value of 0 Volts will cause a Zero to be returned. A value of +5 Volts returns 1024 and a value of 10 Volts returns 2048. Other voltages vary the A/D result proportionally between the two extreme value, of course. The input sensitivity of the A/D System can be altered with the "Gain" Parameter. This parameter reduces the Voltage Range over which the Digitizer's Values are assigned, and, thus, has the effect of increasing the resolution with which the values in the more restricted range are digitized.

The "Schan" Parameter determines the starting input channel number. The "Nchan" Parameter indicates the number of channels, starting with "Schan" that will be sampled on each Scan.

"Npts" determines the number of Scans that will be made by the A/D System. Note that there will be "Npts" Samples for each of the "Nchan" Channels sampled in "Buffer" when ADSET is finished.

The "Rate" and "Count" Parameters, taken together, determine the interval between Scans. The Rate Parameter selects the basic unit for the A/D's Interscan Clock. For Example, a "Rate" Parameter of 3 and a "Count" Parameter of 5 will result in successive scans being separated by 5 Milliseconds.

The "Stat" Variable is Zeroed by the ADSET Call itself. As soon as the digitizer is started, the ADSET Call Returns. The User's Program may go about other business while the Digitizer Sweep is performed. When the complete sweep is finished, the "Stat" Variable is Set to One.

Clinit            - Enable the Hard Clock so that Software Timers can be Used

Clinit -- Enable the Hard Clock so that Software Timers can be Used

Format:

```
CALL clinit
CALL clinit(unit)
CALL clinit(unit,base,count)
```

Where:

```
unit    (Int Val) Number of the Hard Clock Unit to
          use with CLPACK.
base    (Int Val) Basic Rate for the Programmable
          Clock.
count   (Int Val) Number of Units of "Base" to Count
          Off per Each Interrupt.
```

Description:

The CLINIT Subroutine Starts Up the Programmable Clock that services CLPACK's Software Timers. This Subroutine must be executed before any CLSET or CLSTOP

Calls may be executed.

The Zero Argument Form of CLINIT should be used for most applications.

Ckkill            - Turn Off CLPACK's Hardware Programmable Clock

Ckkill -- Turn Off CLPACK's Hardware Programmable Clock

Format:

CALL ckkill

Description:

The CLKILL Subroutine Disables the Programmable Clock that services CLPACK's Software Timers. This has the effect of turning off any Software Timers that might have been active, since, with no Hardware Clock running, they cannot be serviced.

Diinit            - Enable Parallel Inputs

Diinit -- Enable Parallel Inputs

Format:

CALL diinit

CALL diinit(max)

Where:

max        (Int Val) Number of the Highest Bit for which an  
            Input Can Occur.

Description:

The DIINIT Subroutine Enables the Parallel Input Interface. This Subroutine must be executed in order for the DISET, DIVAL, and DIRESP Subroutines to work.

The Optional "Max" Parameter indicates the Highest Bit Number for which inputs are expected.

Dikill            - Turn off Digital Inputs Entirely

Dikill -- Turn off Digital Inputs Entirely...

Format:

CALL dikill

Description:

The DIKILL Subroutine turns off Parallel Inputs,  
thus effectively terminating all digital input recording  
until another DIINIT is done.

DiResp            - Report an input for some bit or other

DiResp -- Report an input for some bit or other

Format:

CALL DiResp(bit,mode,stat)

Where:

bit            (Int Var) Place in which the Bit Position of the latest Digital Input "Hit" is Stored.  
mode          (Int or String Val) Single or Repeat Interrupt Mode:  
                0 or 'Single': Single Interrupt Mode.  
                1 or 'Repeat': Repeat Interrupt Mode.  
stat          (Int Var) Variable that is incremented whenever an input occurs. DiResp Zeroes this Flag Initially.

-OR-

CALL DiResp(bit,mode,type,src,dst,stat)

Where:

bit            (Int Var) Place in which the bit number of the latest digital input "hit" is returned.  
  
mode          (Int or String Value) Single or Repeat interrupt mode:  
                0 or 'Single': Single interrupt mode.  
                1 or 'Repeat': Repeat interrupt mode.  
type          (Int Value) Where to get response time:  
                1: Read Software Timer indicated by "src".  
                2: Read Programmable Clock indicated by "src".  
                3: Copy variable given as "src" when a response occurs.  
src            (Int Value or Variable) Depends on "type" Above.  
dst            (Int Variable) Variable to which the response time for the latest hit, as indicated by "mode" and "src" will be copied. Note that this variable is not zeroed or otherwise changed by the call to DiResp unless an input event occurs. If no input activity takes place, "dst" retains the value it had before the call to DiResp.  
stat          (Int Variable) Variable that is incremented whenever a digital input occurs.

-OR-

CALL diresp()

Description:

The DIRESP subroutine enables a Fortran program to respond to Inputs for any of the Parallel I/O System's Bits. Whenever an Input Bit is Set (by the External Hardware), the

Bit Number on which the Input occurred is copied into "Bit" and One is added to the value of "Flag". "Flag" is Zeroed by the initial call to DIRESP.

The "Mode" Parameter specifies whether DIRESP will stop reporting bits after the first hit, or whether it will continue to report bits until it is disabled.

The "Type", "Src" and "Dst" Parameters allow the User to record a response time along with the bit number. The "Mode" and "Src" Parameters indicates where this response time should be obtained. When a Response occurs, this time is written to "Dst".

If DIRESP is called with no parameters, the effect of any previous DIRESP Call is canceled.

DIINIT must be executed before DIRESP is called in order for DIRESP to work effectively. The execution of a DIINIT or a DIKILL, will (of course) cancel any pending DIRESP.



Diset                - Enable Digital Input Reporting for a Given Bit

Diset -- Enable Digital Input Reporting for a Given Bit.

Format:

CALL diset(bit,mode,stat)

Where:

bit        (Int Var, 0-15) Number of the bit to watch.  
mode       (Int or String Val) Single or Repeat Interrupt Mode:  
          0 or 'Single': Single Interrupt Mode.  
          1 or 'Repeat': Repeat Interrupt Mode.  
stat       (Int Var) Variable that will be up'ed when  
          an input from the Bit indicated by "Bit" is detected.  
          "Stat" is Zero'ed by the Call to DISET.

-OR-

CALL diset(bit,mode,type,src,dst,flag)

Where:

bit        (Int Val, 0-15) Bit Position to Watch.  
mode       (Int or String Val) Single or Repeat Interrupt Mode:  
          0 or 'Single': Single Interrupt Mode.  
          1 or 'Repeat': Repeat Interrupt Mode.  
  
type       (Int Val) Where to get Response Time:  
          1: Read Software Timer indicated by "Src".  
          2: Read Programmable Clock indicated by "Src".  
          3: Copy Var given as "Src" when a response  
             occurs.  
src        (Int Var) Depends on "Mode" Above.  
dst        (Int Var) Variable to which the Response time  
          for the latest hit, as indicated by "Mode"  
          and "Src" will be copied.  
stat       (Int Var) Variable that is Incremented whenever  
          an Input on the indicated bit comes in. This variable  
          is Cleared by the DISET Call Itself.

Description:

The DISET Subroutine enables a User's Fortran Program to respond to activity on a given Parallel Interface Input Bit. When the DISET Subroutine is executed, subsequent inputs for the "Watched" Bit will cause the indicated "Stat" Variable to be incremented. Since each Input will cause "Stat" to increase,

"Stat" will be a count of the number of Inputs that have occurred for the Watched Bit.

The "Mode" Parameter specifies whether DIRESP will stop reporting bits after the first hit, or whether it will continue to report bits until it is disabled.

The "Type", "Src" and "Dst" Parameters allow the

User to record a response time along with the bit number.  
The "Mode" and "Src" Paramters indicates where this response time  
should be obtained. When a Response occurs, this time  
is written to "Dst".

Inputs on the Watched Bit will be processed until either  
a DICLR for the Watched Bit, or a DIINIT or DIKILL is executed.

The DIINIT Subroutine must be executed before DISET can  
be used.

Doclr            - Clear a Specified Output Bit

Doclr — Clear a Specified Output Bit

Format:

CALL doclr(bit)

Where:

bit        (Int Val, 0-15) Bit to Turn Off

Description:

The DOCLR Subroutine Turns Off the Digital Output Bit Designated by the "Bit" Parameter. Only the designated bit is affected. This Subroutine clears only the Bit Indicated.

Doinit            - Clear the Digital Output Register

Doinit — Clear the Digital Output Register

Format:

CALL doinit

Description:

The DOINIT Subroutine Clears the Parallel Interface's Digital Output Register. All 16 Digital Output Bits are Turned Off. It is not necessary to execute a DOINIT to use the other Digital Output Subroutines, but it is often a good idea to make sure that all the Digital Output Bits are Off when starting a program that uses Digital Outputs. A Call to DOINIT is equivalent to a Call to DOKILL or a DOVAL(0).

Dokill - Clear the Digital Output Register

Format:

CALL dokill

Description:

The DOKILL Subroutine Clears All the Digital Output Bits. It is identical to the DOINIT Subroutine. The Two Names are provided for compatibility with other Digital I/O Subroutine Packages. It is not necessary to perform a DOKILL before Exiting or Aborting a Program that has set Output Bits, but, depending on the nature of the equipment connected to the Digital Output System, it is often a good idea.

Doset                - Turn on a Specified Digital Output Bit

Doset -- Turn on a Specified Digital Output Bit

Format:

CALL doset(bit)

Where:

bit        (Int Val, 0-15) Output Bit to Set

Description:

The DOSET Subroutine turns on the Bit designated by the "Bit" Parameter. Bits are Numbered from Right to Left in the Digital Output Register from 0 to 15. This is the conventional way that bit positions are numbered on the PDP11. The designated Output Bit remains set until a DOCLR is executed for it, or until a DOINIT or a DOKILL is Called. The DOINIT Subroutine need not be executed to use DOSET. DOINIT and DOKILL are equivalent to DOCLRs for bits 0 through 15.

GqDisp            - Display a waveform

GqDisp — Display a waveform

Format:

CALL GqDisp(loc,chan,data,nchan,npts,scale)

Where:

loc        (Int Value, 1-12) This parameter gives the waveform location. Waveforms are numbered with waveform 1 at the upper left, waveform 3 at the lower left, waveform 4 at the upper right, and waveform 6 at the lower right. Waveforms 7 through 12 overlay waveforms 1 through 6.

chan       (Int Value) "Channel" from the data array which is to be displayed. This value should be in the range from 1 to nchan.

data       (Int Array, nchan x npts) Array from which the data we will display are to be taken.

nchan      (Int Value) Number of channels in the data array

npts       (Int Value) Number of points in the data array.

scale      (Int Value) Waveform scale factor. Eight is subtracted from this value, and the result is used as follows: If > 0,

each point in the waveform is multiplied by the result.  
If < 0, each point in the waveform is divided by the  
absolute value of the result. If 0, the data are not

scaled.

Description:

The GqDisp subroutine displays a waveform in an indicated display frame on the display created using GqInit. Previously written data are erased first. This subroutine is compatible with GTDISP, except that all six waveforms may be overlayed. Hence, "loc" values 7 through 9 will overlay the waveforms in the left hand column of the display, instead of in the right hand column as with GTPACK. To overlay the right hand column, use "loc" values 10 through 12.

GqIrit            - Initialize the QRGB-Graph Waveform Display

GqInit — Initialize the QRGB-Graph Waveform Display

Format:

CALL GqInit(flag)

Where:

flag      (Int Value) Dummy argument for compatibility with  
            GtPack.

Description:

The GqInit subroutine sets up the Matrox QRGB-Graph waveform display. The "flag" argument does nothing, it is provided for compatibility with GTPACK. The waveform display is always redrawn by GqInit.



GqLab1            - Set a QRGB-Graph display label

GqLab1 -- Set a QRGB-Graph display label

Format:

CALL GqLab1(loc,string)

Where:

loc        (Int Value, 1-12) Label Location.

string    (String) New label string.

Description:

The GqLab1 subroutine resets a Matrox QRGE-Graph waveform display label. To erase a label, an explicit null string (either a "", or a null terminated array) must be given. No zero argument form is allowed. Note that GTPACK allowed a zero argument form to clear a label. Labels are located just below the waveform frame corresponding the the "loc" value given. Labels for "loc" values 1-6 are written above those for values 7-12.

GqTtl                - Set the QRGB-Graph display title line

GqTtl — Set the QRGB-Graph display title line

Format:

CALL GqTtl(string)

Where:

string    (String) New title string.

Description:

The GqTtl subroutine resets the Matrox QRGB-Graph waveform display title line. To erase the title line, an explicit null string (either a "", or a null terminated array) must be given. No zero argument form is allowed. Note that GTPACK allowed a zero argument form to clear the title line.

McChar            - Write a Character on the Matrox

McChar — Write a Character on the Matrox

Format:

CALL mcchar(c)

CALL mcchar(x,y,c)

Where:

c            (Byte Val) Character to write...

x,y        (Int Vals) New X and Y Positions...

Description:

The MCCHAR Subroutine writes a character on the screen of the current size and color at the given screen position. If no position is given, the current cursor position is assumed.

McDisp            - Turn Matrox Display On or Off

McDisp — Turn Matrox Display On or Off

Format:

CALL mcdisp(val)

Where:

val        (Int Val) 0: Turn Off the Matrox, 1: Turn On the Matrox

Description:

The MCDISP Subroutine turns the Matrox On or Off.

McEras            - Make an Area the Background Color

McEras — Make an Area the Background Color

Format:

CALL mceras  
CALL mceras(x,y)  
CALL mceras(a,b,x,y)

Where:

a,b        (Int Vals) Coordinates of Erase Area Upper Left Hand Corner.  
x,y        (Int Vals) Coordinates of Erase Area Lower Right Hand Corner.

Description:

This MCERAS Subroutine sets an area of the Matrox Screen to the current background color. If no coordinates are given, a hardware flood is performed using the current background color. If two coordinates are given, the rectangle determined by the current position on the upper left, and the given point on the lower right will be colored using the current background color. If four parameters are given, the first two will designate the upper left hand corner of the area to erase, and the next two will designate the lower right hand corner of the area.

McInit            - Initialize the Matrox

McInit -- Initialize the Matrox

Format:

CALL mcinit  
CALL mcinit(flag)

Where:

flag      (Int Val) 0: Don't Erase. 1: Erase.

Description:

The MCINIT Subroutine initializes the Matrox. It should be called before any other MCPACK Subroutines are called.

If a "Flag" value is given, the screen is erased or not erased as indicated. If no value is given, the screen is not erased. Leaving the screen unerased by MCINIT allows programs to generate Matrox images and chain to other programs without losing them.

MCINIT sets defaults as follows:

Foreground:	256, or White.
Background:	0, or Black.
Spacing:	Nonproportional.
Position:	(0,0)
Origin:	(0,0)
Scroll:	0
Video:	On
Reverse:	Off

If an initial erase is not done, some of these default values will not be established until a subsequent MCPACK Subroutine accesses the Matrox.

McSync            - Wait for Blanking to Start

McSync — Wait for Blanking to Start

Format:

CALL mcsync

Description:

The MCSYNC Subroutine waits until the Matrox's Vertical Blanking Bit goes from Off to On.

Mctext            - Write a Null Terminated String on the Matrox

Mctext -- Write a Null Terminated String on the Matrox

Format:

CALL mctext(string)

CALL mctext(x,y,string)

Where:

x,y        (Int Vals) Place to put the cursor at first

string    (Byte Array, or Whatnot) A null terminated  
          buffer filled with Ascii Characters.

Description:

The MCTEXT writes a Null Terminated String of Characters out on the Matrox in the current Color and Size. If no position is specified, the current cursor position is used.



Pcinit            - Initialize a Programmable Clock

Pcinit — Initialize a Programmable Clock

Format:

CALL pcinit(clock)

Where:

clock    (Int Val, 1-6) Clock to Set Up.

         - OR -

i=ipcini()

Where:

i        (Int Result) Clock Allocated (1-6, or 0 if None are  
         available.)

Description:

The PCINIT Subroutine initializes the indicated Programmable Clock. This Subroutine must be called before any given clock can be used. "Clock" specifies which clock should be initialized. The given clock is marked as busy until a PCKILL is performed for it. Attempts to Initialize it in the mean time will fail.

The IPCINI Function returns the number of an available Programmable Clock, and initializes that clock. If no clocks are available, Zero is Returned.

Pckill                - Turn Off and Free One or All Clocks

Pckill — Turn Off and Free One or All Clocks

Format:

CALL pckill  
CALL pckill(clock)

Where:

clock    (Int Val, 1-6) Clock to Kill and Free

Description:

The PCKILL Subroutine Turns Off and Frees One or all of the Programmable Clocks. If "Clock" is specified, only the Indicated Clock is Killed. If "Clock" is not specified, all the Clocks are Killed.

Pcset            - Turn On a Programmable Clock

Pcset -- Turn On a Programmable Clock

Format:

```
CALL pcset(clock,mode,rate,count)
CALL pcset(clock,mode,rate,count,stat)
```

Where:

clock    (Int Val, 1-6) Clock to Use.  
mode    (Int Val) Single or Repeat Mode.  
         0 or 'Single': Single Sweep Mode  
         1 or 'Repeat': Repeat Sweep Mode  
rate    (Int Val, 0-7) Basic Clock Rate:

0:	1	Microsecond	1	Megahertz
1:	10	Microseconds	100	Kilohertz
2:	100	Microseconds	10	Kilohertz
3:	1	Millisecond	1	Kilohertz
4:	10	Milliseconds	100	Hertz
5:	100	Milliseconds	10	Hertz
6:	1	Second	1	Hertz
7:	10	Seconds	100	Millihertz

count    (Int Val, 0-32767) Number of Units of "Rate" per  
         Interrupt. (0 means 32768 Units.)  
stat    (Int Var) Status Variable. This is Zeroed when  
         PCSET is Called. One is Added to "Stat" every  
         time a Clock Sweep Completes.

Description:

The PCSET Subroutine starts one of Pearl II's programmable clocks. Clocks may be run either in "Single Sweep Mode", in which case one interval of the specified rate is timed, or in "Repeat Sweep Mode", in which the clock times intervals indefinitely until stopped.

The "Clock" Parameter specifies which of Pearl II's programmable clocks should be used.

The "Mode" Parameter indicates whether the Clock should run in Single or Repeat Sweep Mode.

The "Rate" Parameter specifies the Basic Rate at which the Clock will run. "Count" units at the given rate will be timed for each sweep.

The "Stat" Variable is zeroed by the call to PCSET. One is added to this Variable every time a Clock Sweep is completed.

If "Stat" is not given, the clock is run without interrupts.

If a timer is running when this subroutine is called, it is turned off first.

Wait                    - Wait for a Variable to Become Non-Zero

Wait -- Wait for a Variable to Become Non-Zero

Format:

CALL wait(v1)

CALL wait(v1,v2,...,vn)

Where:

v1...vn (Int Vars) The Variables to Test for Non-Zero Values

Description:

The WAIT Subroutine Tests the Values of All the Variables Passed to it. As soon as ANY of these variables becomes something other than Zero, the Subroutine Returns to the User's Program. The typical application for this Subroutine is testing the "Stat" Variables that keep the user informed of the progress of various other LABPAK Subroutine initiated activity.

Note:

The WAIT Subroutine is implemented using the PDP11's WAIT Instruction. This instruction causes the Central Processor to relinquish control of the Bus to those DMA Devices that might be competing for it until an Interrupt Request has been serviced. The Variable list is first tested when this Subroutine is called, and if the contents of all the variables are zero, a WAIT instruction is executed. When this instruction falls through, (meaning that an interrupt service routine that might have changed the value of one of the variables was executed) the tests are performed again. When One of the Variables becomes Non-Zero, the Subroutine falls through.

Zero                - Array Zeroing Subroutine

Zero -- Array Zeroing Subroutine

Format:

CALL zero(buf,size)

Where:

Buf            (Array, "Size" Bytes in Size) The Array  
                 to Fill with Zeroes.

Size          (Int Val) The Size, in Bytes, of the "Buf" Array.

Description:

The ZERO Subroutine Sets every element of the indicated Array to Zero. It does this in a manner more efficient than the corresponding DO Loop Fortran Code could do it, and is more concise. The Size in Bytes of an Array Variable may be calculated by multiplying the sizes of the Array's Dimensions together and multiplying this result by the size of an individual element of the Array.

Example:

The following short Fortran Program illustrates the use of the ZERO Subroutine:

```
PROGRAM zero
C
C Declare Some Arrays Here...
C
      INTEGER dog(2,4,6)      !2*4*6*2 = 96 Bytes
      REAL mouse(100,2)      !100*2*4 = 800 Bytes
C
C Zero these and Exit...
C
      CALL zero(dog,2*4*6*2)   !Dog is all Zeroes Now...
      CALL zero(cat,100*2*4)   !Cat is Zeroed, too...
      STOP 'Dog and Cat Zeroed...' !Say what this did...
      END
```

## Appendix K

F.PEAK Program to perform single trial data analysis on ERPs collected by PMSM. Also generates are average waveforms for each experimental condition. This program creates unadjusted and latency adjusted waveforms, and single trial estimates of P300 latency and amplitude. Four different algorithms are included: Vector cross correlation, Pz Cross correlation, Vector peak picking, and Pz peak picking.

```

C*****
C
C          F. PEAK
C
C          PROGRAMMER: DAVID STRAYER
C          UNIVERSITY OF ILLINOIS
C
C*****
C
C      IDS USED AND GENERATED BY THIS PROGRAM
C
C      ID(1)=SUBJECT
C      ID(2)=SESSION
C      ID(3)=BLOCK
C      ID(4)=EEG CHANNEL (EOG, FZ, CZ, PZ, STM)
C      ID(5)=DELAY
C      ID(6)=SOA
C      ID(7)=MEMSIZ
C      ID(8)=CMVM
C      ID(9)=REACTION TIME
C      ID(10)=RESPONSE CLASSIFICATION (0=NO RESPONSE, 1=HIT, 2=CR,
C          3=FA,4=MISS)
C
C      ID'S INCLUDED IN THE OUTPUT OF THIS PROGRAM
C
C      SINGLE TRIAL ESTIMATES OF P300 TO S1 (300 TO 600 MSEC)
C
C      ID(11)=LATENCY OF CROSS CORR P300 (COMPOSITE)
C      ID(12)=AMPLITUDE OF CROSS CORR P300 (COMPOSITE)
C      ID(13)=LATENCY OF CROSS CORR P300 (PZ CHANNEL)
C      ID(14)=AMPLITUDE OF CROSS CORR P300 (PZ CHANNEL)
C      ID(15)=LATENCY OF PEAKPICKED P300 (COMPOSITE)
C      ID(16)=AMPLITUDE OF PEAKPICKED P300 (COMPOSITE)
C      ID(17)=LATENCY OF PEAK PICKED P300 (PZ CHANNEL)
C      ID(18)=AMPLITUDE OF PEAK PICKED P300 (PZ CHANNEL)
C
C      SINGLE TRIAL ESTIMATES OF P300 TO S2 (800 TO 1700 MSEC)
C
C      ID(19)=LATENCY OF CROSS CORR P300 (COMPOSITE)
C      ID(20)=AMPLITUDE OF CROSS CORR P300 (COMPOSITE)
C      ID(21)=LATENCY OF CROSS CORR P300 (PZ CHANNEL)
C      ID(22)=AMPLITUDE OF CROSS CORR P300 (PZ CHANNEL)
C      ID(23)=LATENCY OF PEAKPICKED P300 (COMPOSITE)
C      ID(24)=AMPLITUDE OF PEAKPICKED P300 (COMPOSITE)
C      ID(25)=LATENCY OF PEAK PICKED P300 (PZ CHANNEL)
C      ID(26)=AMPLITUDE OF PEAK PICKED P300 (PZ CHANNEL)
C
C      REAL ID(26),DATA(210,5),YDATA(210,5),
C      +FZ(210),CZ(210),PZ(210),P3(210),BASE(3),TEMP(50),
C      +NTRIAL(24,5),AVEID(26,24,5),AVE(210,3,24,5),BAVE(24,5)
C

```



```

C      INTEGER FNAME(6),LOW(2),HIGH(2),BIN,IDS(108)
C
C      CALL OPEN(3,ISTAT)
C
C      DEFINE WINDOW ONSET AND OFFSET
C
C      DATA LOW(1),HIGH(1)/40,70/  ! FOR S1
C      DATA LOW(2),HIGH(2)/90,180/  ! FOR S2
C
C      ASSIGN FILES
C
C      CALL ASARG(1, 20,"SUB          ",FNAME)
C      CALL ASARG(2, 30,"DATA        ",FNAME)
C      CALL ASARG(3,-40,"PK-OUT      ",FNAME)
C      CALL ASARG(4,-41,"ADJ-OUT     ",FNAME)
C
C      DEFINE TEMPLET (POSITIVE SEGMENT OF 0.5 HZ SINE WAVE)
C
C      FOR I=1,50
C      TEMP(I)=-COS(I*6.2829/50)
C      END FOR
C
C      SET UP FILTER PARAMETERS SEE RUTCHKIN AND GLASER (1979)
C
C      TT=10.0          ! SAMPLING INTERVAL
C      NPTS=210         ! NUMBER OF POINTS IN WAVEFORM TO FILTER
C      LGL=6            ! LENGTH OF LOW PASS FILTER
C      NUML=2           ! NUMBER OF ITERATIONS OF LOW PASS FILTER
C      LGH=0            ! LENGTH OF HIGH PASS FILTER
C      NUMH=0           ! NUMBER OF ITERATIONS OF HIGH PASS FILTER
C      NSTART=1+(LGL*NUML)+(LGH*NUMH)
C      NSTOP=NPTS-(LGL*NUML+LGH*NUMH)
C      CALL FILTN(1,NPTS,NSTART,NSTOP,1,LGL,NUML,LGH,NUMH)
C
C      DESCRIBE FILTER PARAMETERS
C
C      FOL=1000.0/((2*LGL+1)*TT)
C      IF(NUML.EQ.1) F3DB=.44*FOL
C      IF(NUML.EQ.2) F3DB=.31*FOL
C      WRITE(3,*) F3DB,"Hz (-3dB) low pass filter"
C
C      READ IN A TRIAL
C
C      LOOP
C
C      FOR J=1,5
C      READ(30,100,END=50) (ID(K),K=1,10),(DATA(I,J),I=1,210)
C      END FOR
C
C      FIND THIS TRIAL IN THE SUB ID FILE
C

```

```

1  READ(20,101) IDS
   IF(ID(1).NE.IDS(1).OR.ID(2).NE.IDS(2).OR.ID(3).NE.IDS(3)) GOTO 1
   DELAY=IDS(4)
C
C  DETERMINE NUMBER OF CORRECT RUNNING MEMORY TRIALS
C
   ICORR=0
   IF(DELAY.GT.2) THEN
   FOR I=3,ABS(DELAY)
   IF(IDS(I*5+7).EQ.IDS(I*5+8)) ICORR=ICORR+1
   END FOR
   END IF
C
   IF(ICORR.GE.((DELAY-2.0)*0.50)) THEN ! ABOVE CHANCE ACCURACY
C
   FOR EPOCH=1,2
C
   COMPUTE BASELINE (AVERAGE OF 100 MSEC EPOCH)
   S1 EPOCH BASELINE: -100 MSEC TO 0 MSEC
   S2 EPOCH BASELINE: 500 MSEC TO 600 MSEC
C
   FOR J=1,3
   BASE(J)=0.
   FOR K=1+((EPOCH-1)*59),10+((EPOCH-1)*59)
   BASE(J)=BASE(J)+(DATA(K,(J+1))/10.0)
   END FOR
   END FOR
C
C  SUBTRACT BASELINE
C
   FOR IPT=1,210
   FZ(IPT)=DATA(IPT,2)-BASE(1)
   CZ(IPT)=DATA(IPT,3)-BASE(2)
   PZ(IPT)=DATA(IPT,4)-BASE(3)
   END FOR
C
C  FILTER THE EEG DATA
C
   CALL FILTR(FZ,FZ)
   CALL FILTR(CZ,CZ)
   CALL FILTR(PZ,PZ)
C
C  COMPUTE THE P300 VECTOR FILTER
C
   CALL VECT(210,300.,15.,FZ,CZ,PZ,P3)
C
C  COMPUTE CROSS CORRELATION ON P300 VECTOR (COMPOSITE)
C
   CALL CROSS(210,LOW(EPOCH),HIGH(EPOCH),TEMP,P3,CORP3,LAT,AMP)
   ID(3+(8*EPOCH))=LAT
   ID(4+(8*EPOCH))=AMP

```

```

      IF(CORP3.LE..30) ID(3+(8*EPOCH))=0.0
C
C   COMPUTE CORSS CORRELATION ON PZ CHANNEL
C
      CALL CROSS(210,LOW(EPOCH),HIGH(EPOCH),TEMP,PZ,CORPZ,LAT,AMP)
      ID(5+(8*EPOCH))=LAT
      ID(6+(8*EPOCH))=AMP
      IF(CORPZ.LE..30) ID(5+(8*EPOCH))=0.0
C
C   PEAK PICK ON P300 VECTOR FILTER OUTPUT
C
      CALL PKPICK(P3,210,LOW(EPOCH),HIGH(EPOCH),LAT,AMP)
      ID(7+(8*EPOCH))=LAT
      ID(8+(8*EPOCH))=AMP
      IF(ID(7+(8*EPOCH)).LE.(LOW(EPOCH)*10-100).OR.ID(7+(8*EPOCH)).GE.
      +(HIGH(EPOCH)*10-100)) ID(7+(8*EPOCH))=0.0
C
C   PEAK PICK ON PZ CHANNEL
C
      CALL PKPICK(PZ,210,LOW(EPOCH),HIGH(EPOCH),LAT,AMP)
      ID(9+(8*EPOCH))=LAT
      ID(10+(8*EPOCH))=AMP
      IF(ID(9+(8*EPOCH)).LE.(LOW(EPOCH)*10-100).OR.ID(9+(8*EPOCH)).GE.
      +(HIGH(EPOCH)*10-100)) ID(9+(8*EPOCH))=0.0
C
      END FOR
C
C   WRITE OUT ID FILE WITH PEAK ESTIMATES
C
      WRITE(40,102) ID
C
C   NOW LATENCY ADJUST EACH TRIAL
C
      FOR EST=1,5
C
C   CONVERT P3 LATENCY TO NUMBER OF DATA POINTS
C
      IF(EST.NE.5) THEN
      IF(ID((EST*2)+17).EQ.0) GO TO 40
      NPOINT=(ID((EST*2)+17)+100)/10
      ELSE
      NPOINT=120      ! EST 5 IS THE UNADJUSTED AVERAGE
      END IF
C
C   WHICH WAY SHOULD WE SHIFT THE WAVEFORM?
C
      IF(NPOINT.LT.120) ISHIFT=-1
      IF(NPOINT.EQ.120) ISHIFT=0
      IF(NPOINT.GT.120) ISHIFT=1
C
C   NOW SHIFT THE WAVEFORM

```

```

C      IF(ISHIFT.EQ.0) THEN
        FOR K=2,4
        FOR J=1,210
        YDATA(J,K)=DATA(J,K)
        END FOR
        END FOR
        GO TO 150

C
C      ELSE IF(ISHIFT.EQ.-1) THEN
C
C      SHIFT THIS WAVEFORM TO THE RIGHT ----->
C
        NDIFF=120-NPOINT
        JMOVE=210-NDIFF
        FOR K=2,4
        XBASE=DATA(1,K)
        FOR J=1,JMOVE
        YDATA(J+NDIFF,K)=DATA(J,K)
        END FOR

C
C      NOW SET THE TRUNCATED REGION TO THE FIRST DATA POINT
C
        FOR J=1,NDIFF
        YDATA(J,K)=XBASE
        END FOR
        END FOR
        GO TO 150

C
C      ELSE IF(ISHIFT.EQ.1) THEN
C
C      SHIFT THIS WAVEFORM TO THE LEFT <-----
C
        NDIFF=NPOINT-120
        JMOVE=210-NDIFF
        FOR K=2,4
        XBASE=DATA(210,K)
        FOR J=1,JMOVE
        YDATA(J,K)=DATA(J+NDIFF,K)
        END FOR

C
C      NOW SET THE TRUNCATED REGION TO THE LAST DATA POINT
C
        ITEMP=210-NDIFF+1
        FOR J=ITEMP,210
        YDATA(J,K)=XBASE
        END FOR
        END FOR
        GO TO 150
        END IF
C

```

```

150  CONTINUE
C
C  IDENTIFY WHICH BIN TRIAL SHOULD BE ASSIGNED TO
C
    BIN=0
C
    IF(ID(10).EQ.1) THEN ! TARGETS
C
    IF(ID(8).EQ.1) THEN ! CM CONDITIONS
C
    IF(ID(5).EQ.0.AND.ID(7).EQ.2) BIN=1
    IF(ID(5).EQ.4.AND.ID(7).EQ.2) BIN=2
    IF(ID(5).EQ.15.AND.ID(7).EQ.2) BIN=3
    IF(ID(5).EQ.0.AND.ID(7).EQ.4) BIN=4
    IF(ID(5).EQ.4.AND.ID(7).EQ.4) BIN=5
    IF(ID(5).EQ.15.AND.ID(7).EQ.4) BIN=6
C
    ELSE IF(ID(8).EQ.2) THEN ! VM CONDITIONS
C
    IF(ID(5).EQ.0.AND.ID(7).EQ.2) BIN=7
    IF(ID(5).EQ.4.AND.ID(7).EQ.2) BIN=8
    IF(ID(5).EQ.15.AND.ID(7).EQ.2) BIN=9
    IF(ID(5).EQ.0.AND.ID(7).EQ.4) BIN=10
    IF(ID(5).EQ.4.AND.ID(7).EQ.4) BIN=11
    IF(ID(5).EQ.15.AND.ID(7).EQ.4) BIN=12
    END IF
C
    ELSE IF(ID(10).EQ.2) THEN ! DISTRACTORS
C
    IF(ID(8).EQ.1) THEN ! CM CONDITIONS
C
    IF(ID(5).EQ.0.AND.ID(7).EQ.2) BIN=13
    IF(ID(5).EQ.4.AND.ID(7).EQ.2) BIN=14
    IF(ID(5).EQ.15.AND.ID(7).EQ.2) BIN=15
    IF(ID(5).EQ.0.AND.ID(7).EQ.4) BIN=16
    IF(ID(5).EQ.4.AND.ID(7).EQ.4) BIN=17
    IF(ID(5).EQ.15.AND.ID(7).EQ.4) BIN=18
C
    ELSE IF(ID(8).EQ.2) THEN ! VM CONDITIONS
C
    IF(ID(5).EQ.0.AND.ID(7).EQ.2) BIN=19
    IF(ID(5).EQ.4.AND.ID(7).EQ.2) BIN=20
    IF(ID(5).EQ.15.AND.ID(7).EQ.2) BIN=21
    IF(ID(5).EQ.0.AND.ID(7).EQ.4) BIN=22
    IF(ID(5).EQ.4.AND.ID(7).EQ.4) BIN=23
    IF(ID(5).EQ.15.AND.ID(7).EQ.4) BIN=24
C
    END IF
C
    END IF
    IF(BIN.EQ.0) GO TO 40

```

```

C
C      SUM THE POINTS

      NTRIAL(BIN,EST)=NTRIAL(BIN,EST)+1
      FOR KID=1,26
      AVEID(KID,BIN,EST)=ID(KID)
      END FOR
      FOR LCHAN=2,4
      FOR LPT=1,210
      AVE(LPT,LCHAN-1,BIN,EST)=
+AVE(LPT,LCHAN-1,BIN,EST)+YDATA(LPT,LCHAN)
      END FOR
      END FOR

C
      END FOR
      END IF
40    END LOOP
50    CONTINUE
C
C      CALCULATE AVERAGES FOR EACH CONDITION
C
      FOR EST=1,5
      DO 300 MBIN=1,24
      IF(NTRIAL(MBIN,EST).EQ.0) GO TO 300
      FOR MCHAN=1,3
      FOR MPT=1,210
      AVE(MPT,MCHAN,MBIN,EST)=AVE(MPT,MCHAN,MBIN,EST)/NTRIAL(MBIN,EST)
      IF(MPT.LE.10) BAVE(MCHAN,MBIN,EST)=BAVE(MCHAN,MBIN,EST)
      *+(AVE(MPT,MCHAN,MBIN,EST)/10)
      END FOR
      END FOR
300    CONTINUE

C      SUBTRACT BASELINE FROM AVERAGE AND OUTPUT AVERAGES

      FOR JBIN=1,24
      FOR JCHAN=1,3
      FOR JPT=1,210
      AVE(JPT,JCHAN,JBIN,EST)=AVE(JPT,JCHAN,JBIN,EST)
      *-BAVE(JCHAN,JBIN,EST)
      END FOR
      WRITE(41,100) EST,REAL(JBIN),REAL(JCHAN),REAL(NTRIAL(JBIN,EST)),
      +(AVEID(K,JBIN,EST),K=1,26),(AVE(N,JCHAN,JBIN,EST),N=1,210)
      END FOR
      END FOR
      END FOR
      STOP
100    FORMAT(22F6.0)
101    FORMAT(16I5)
102    FORMAT(11F6.0)
      END

```

```

C
C      SUBROUTINE VECT(NPT,OR,POL,FZ,CZ,PZ,P3)
C
C      THIS SUBROUTINE PERFORMS A VECTOR FILTER
C      FOR ANY GIVEN POLARITY AND ORIENTATION.
C      THE DATA MUST COME FROM THREE SCALP ELECTRODES.
C
C      ARGUMENTS:
C          NPT          # OF POINTS (MAX=700)
C          OR           ORIENTATION ANGLE
C                     FOR P300 USE OR=300 OR 330
C          POL          POLARITY ANGLE
C                     FOR P300 USE POL=15 OR 30
C          FZ          (210) DATA ARRAYS. IT IMPLIES:
C          CZ          (210)
C          PZ          (210)
C
C                     EL1=FZ, EL2=CZ, EL3=PZ
C          P3          (210) VECTOR FILTERED DATA
C
C      DIMENSION FZ(NPT),CZ(NPT),PZ(NPT),P3(NPT)
C
C      VLOAD=COS(POL*6.2829/360.)
C      XLOAD=COS(OR*6.2829/360.)*VLOAD
C      YLOAD=SIN(OR*6.2829/360.)*VLOAD
C      ZLOAD=SIN(POL*6.2829/360.)
C
C      FOR K=1,NPT
C  C ROTATE AXES
C          Z=(FZ(K)+CZ(K)+PZ(K))* .57735
C          Y=(2.*CZ(K)-(FZ(K)+PZ(K)))* .40825
C          X=(PZ(K)-FZ(K))* .70711
C  C VECTOR FILTER
C          P3(K)=X*XLOAD+Y*YLOAD+Z*ZLOAD
C      END FOR
C      RETURN
C      END

```

```

C
C SUBROUTINE CROSS
C
      SUBROUTINE CROSS(NPTS,ILOW,IHIGH,TEMP,DATA,CORMAX,LAT,BMAX)
      DIMENSION DATA(NPTS),TEMP(50),WIND(50)
      LBEG=ILOW
      LEND=IHIGH
      IF(LBEG.LT.25)LBEG=25
      IF(LEND.GT.(NPTS-25)) LEND=NPTS-25
      CORMAX=0.
      FOR LAG=LBEG,LEND
        IBEG=LAG-25
        IEND=LAG+24
        FOR I=IBEG,IEND
          WIND(I-IBEG+1)=DATA(I)
        END FOR
        CALL CORR(WIND,TEMP,50,R,B)
        IF(R.GE.CORMAX)THEN
          CORMAX=R
          BMAX=B
          LAT=LAG*10-100
        END IF
      END FOR
      RETURN
      END

```



```

C
C      SUBROUTINE CORR
C
      SUBROUTINE CORR(X,Y,NPTS,R,B)
      DIMENSION X(NPTS),Y(NPTS)
      R=0.
      IF(NPTS.LT.2)RETURN
      XSUM=0.
      YSUM=0.
      X2SUM=0.
      Y2SUM=0.
      XYSUM=0.
      FOR I=1,NPTS
        XSUM=X(I)+XSUM
        YSUM=Y(I)+YSUM
        X2SUM=X(I)*X(I)+X2SUM
        Y2SUM=Y(I)*Y(I)+Y2SUM
        XYSUM=X(I)*Y(I)+XYSUM
      END FOR
      XMEAN=XSUM/NPTS
      YMEAN=YSUM/NPTS
      XVAR=X2SUM/NPTS-XMEAN*XMEAN
      YVAR=Y2SUM/NPTS-YMEAN*YMEAN
      COV=XYSUM/NPTS-XMEAN*YMEAN
      IF(XVAR.LE.0..OR.YVAR.LE.0.)RETURN
      XSD=SQRT(XVAR)
      YSD=SQRT(YVAR)
      R=COV/(XSD*YSD)
      B=R*XSD/YSD
      RETURN
      END

```

```

C
C  SUBROUTINE PKPICK
C
SUBROUTINE PKPICK (VECTOR,NPTS,ILOW,IHIGH,LAT,AMP)
DIMENSION VECTOR(NPTS)
PEAK=VECTOR(ILOW)
LAT=ILOW
FOR J=ILOW,IHIGH
IF(VECTOR(J).LE.PEAK) GO TO 10
PEAK=VECTOR(J)
LAT=J
10  END FOR
LAT=LAT*10-100
AMP=PEAK
RETURN
END

```

```

C    FILTER SUBROUTINES TAKEN FROM CPL LIBRARY LIB*F.TSLIB
C
C    SUBROUTINE FILTR(WAV1,WAV2),FILTN(NEW,LTIME,NSTRT,
INSTOP,NTYPE,LGL,NUML,LGH,NUMH)
C    LOW PASS AND HIGH PASS FILTER IN CASCADE
C    WAV1=INPUT; WAV2=OUTPUT
C    PROGRAMMER - D.S.RUCHKIN
C    VERSION - 5/25/75
C    DIMENSION WAV1(LTIME),WAV2(LTIME),TWAV(2001)
C    INTEGER HSTRT,HSTT,HSTOP,HSTP,HRAN
C
C    IF(NTYPE.EQ.0) RETURN
C    IF(LGH.EQ.0.AND.LGL.EQ.0) RETURN
C    IF(LGH.NE.0) GO TO 20
C
C    ONLY LOW PASS
C
C    CALL BOX(LSTRT,LSTOP,LGL,NUML,DIVL,WAV1,WAV2,LTIME)
C    CALL EDGE(NSTT1,NSTP1,LTIME,WAV1,WAV2)
C    RETURN
C
C    HIGH OR BAND PASS
C
C    20 CALL BOX(HSTRT,HSTOP,LGH,NUMH,DIVH,WAV1,TWAV,LTIME)
C    DO 25 K=HSTRT,HSTOP
C    25 WAV2(K)=WAV1(K)-TWAV(K)
C    IF(LGL.NE.0) GO TO 50
C
C    ONLY HIGH PASS
C
C    40 CALL EDGE(NST1,NSTP1,LTIME,WAV1,WAV2)
C    RETURN
C
C    BAND PASS
C
C    50 CALL BOX(LSTRT,LSTOP,LGL,NUML,DIVL,WAV2,WAV2,LTIME)
C    CALL EDGE(NSTT1,NSTP1,LTIME,WAV1,WAV2)
C    RETURN
C
C    INITIALIZATION ENTRY
C
C    ENTRY FILTN
C    IF NEW=0; RETAIN OLD PARAMETERS
C    IF NEW=1; READ NEW PARAMETERS
C    IF(NEW.EQ.0) GO TO 280
C    NSTT1=NSTRT-1
C    NSTP1=NSTOP+1
C    IF(LGL.EQ.0) GO TO 120
C
C    LOW PASS PARAMETERS
C

```

```

        DIVL=2*LGL+1
        DIVL=1.0/DIVL
        LRAN=(NUML-1)*LGL
        LSTRT=NSTRT-LRAN
        LSTT=LSTRT-LGL
        LSTOP=NSTOP+LRAN
        LSTP=LSTOP+LGL
        IF(LSTT.LT.1.OR.LSTP.GT.LTIME) GO TO 250
C
C    HIGH PASS PARAMETERS
C
120    IF(LGH.EQ.0) GO TO 200
        DIVH=2*LGH+1
        DIVH=1.0/DIVH
        IF(LGL.EQ.0) GO TO 140
C
C    BAND PASS-HIGH PARAMETERS
C
        HRAN=(NUMH-1)*LGH
        HSTRT=LSTT-HRAN
        HSTOP=LSTP+HRAN
        GO TO 145
C
C    ONLY HIGH PASS
C
140    HRAN=(NUMH-1)*LGH
        HSTRT=NSTRT-HRAN
        HSTOP=NSTOP+HRAN
145    HSTT=HSTRT-LGH
        HSTP=HSTOP+LGH
        IF(HSTT.LT.1.OR.HSTP.GT.LTIME) GO TO 250
C
C    OUTPUT AND RETURN
C
200    WRITE(6,501) NTYPE,NSTRT,NSTOP,LGL,NUML,LGH,NUMH
        IF(LGL.NE.0) WRITE(6,502) LSTRT,LSTOP,DIVL
        IF(LGH.NE.0) WRITE(6,503) HSTRT,HSTOP,DIVH
        RETURN
C
C    ERROR
C
250    NTYPE=0
        WRITE(6,504)
        GO TO 200
C
C    SAME PARAMETERS
C
280    IF(NTYPE.EQ.0) GO TO 200
        WRITE(6,505)
        RETURN
C

```

```

C      FORMATS
C
500    FORMAT(I2,6I4)
501    FORMAT('      FILTR,5/25/75,NTYPE='I2/,5X,'NSTRT='I4,
1      '      NSTOP='I4,'    LGL='I4,'    NUML='I4,'    LGH='I4,
2      '      NUMH='I4)
502    FORMAT(5X,'LSTRT='I4,'    LSTOP='I4,'    DIVL='F8.5)
503    FORMAT(5X,'HSTRT='I4,'    HSTOP='I4,'    DIVH='F8.5)
504    FORMAT('      FILTER ERROR')
505    FORMAT('      NO FILTER',/)
      END

```

```

C
C
C
C
C
      SUBROUTINE BOX(NSTRT,NSTOP,LG,NUMH,DIV,WAVE1,TWAV1,NPOINT)
C
C      ITERATIVE BOX-CAR FILTER
C      WAVE1=INPUT; TWAVE=OUTPUT
C
      DIMENSION WAVE1(NPOINT),TWAV1(NPOINT),WAVE3(2001),TWAVE(2001)
C
C      TRANSMIT INPUT TO WORK AREA
C
      NCL=NSTRT-LG
      NCU=NSTOP+LG
      DO 4 J=NCL,NCU
4      WAVE3(J)=WAVE1(J)
      NUMIT=1
      MSTRT=NSTRT
      MSTOP=NSTOP
C
C      BEGIN ITERATION LOOP
C
10     DO 50 J=MSTRT,MSTOP
      SUM=0.0
      NCL=J-LG
      NCU=J+LG
      DO 40 K=NCL,NCU
40     SUM=SUM+WAVE3(K)
50     TWAVE(J)=SUM*DIV
      IF(NUMIT.GE.NUMH) GO TO 100
      NUMIT=NUMIT+1
      DO 5 J=MSTRT,MSTOP
5      WAVE3(J)=TWAVE(J)
      MSTRT=MSTRT+LG
      MSTOP=MSTOP-LG
      GO TO 10
C
C      MOVE OUTPUT AND RETURN
C
100    DO 110 K=NSTRT,NSTOP
110    TWAV1(K)=TWAVE(K)
C MODIFICATION BY E. HEFFLEY 9/13/77
C MODIFIED MOVING AVERAGE DONE AT BEGINNING
C AND END OF EPOCH. THIS IS FOR CASES WHEN ITERATIONS=1.
C IT AVERAGES ACROSS LESS THAN 2*LG+1 POINTS BY REDISTRIBUTING
C WEIGHTS ACROSS POINTS AT BEGINNING (OR END) OF EPOCH
C SUCH THAT POINTS CLOSER TO THE BEGINNING ARE GIVEN HIGHER
C WEIGHTS BECAUSE PRESUMABLY THEY ARE BETTER PREDICTORS OF
C THE PRECEDING POINTS (WHICH WERE NOT RECORDED BUT SHOULD HAVE

```

```

C BEEN TO ALLOW FILTERING OF FIRST LG POINTS IN THE EPOCH).
  BASE=DIV
  ISTART=NSTOP+1
  DO 200 IPT=ISTART,NPOINT
    NLEFT=(LG+1)+(NPOINT-IPT)
    NREMNI=(2*LG+1)-NLEFT
    AREA=NREMNI*BASE
    DX=NLEFT
    DY=(2.0*AREA)/DX
    SLOPE=DY/(DX-1.0)
    IFIRST=IPT-LG
    SUM=0.0
    WTOT=0.0
    DO 160 IPTR=IFIRST,NPOINT
      WEIGHT=(IPTR-IFIRST)*SLOPE
      WTOT=WTOT+WEIGHT
      SUM=SUM+WAVE1(IPTR)*(WEIGHT+BASE)
160  CONTINUE
      TWAV1(IPT)=SUM
C  WRITE(6,1200) ISTART,NLEFT,NREMNI,AREA,DX,DY,SLOPE,WTOT
1200  FORMAT(' START=',I4,' LEFT=',I4,' NREMNI=',I4,' /,
1  ' AREA=',F8.4,' DX=',F8.4,' DY=',F8.4,' SLOPE=',F8.4,
2  ', WTOT=',F8.4)
200  CONTINUE
      IEND=NSTRT-1
      DO 300 IPT=IEND,1,-1
        NLEFT=(LG+1)+(IPT-1)
        NREMNI=(2*LG+1)-NLEFT
        AREA=NREMNI*BASE
        DX=NLEFT
        DY=(2.0*AREA)/DX
        SLOPE=DY/(DX-1.0)
        ILAST=IPT+LG
        SUM=0.0
        WTOT=0.0
        DO 260 IPTR=ILAST,1,-1
          WEIGHT=(ILAST-IPTR)*SLOPE
          WTOT=WTOT+WEIGHT
          SUM=SUM+WAVE1(IPTR)*(WEIGHT+BASE)
260  CONTINUE
          TWAV1(IPT)=SUM
C  WRITE(6,1220) IEND,NLEFT,NREMNI,AREA,DX,DY,SLOPE,WTOT
1220  FORMAT(' END=',I4,' LEFT=',I4,' NREMNI=',I4,' /,
1  ' AREA=',F8.4,' DX=',F8.4,' DY=',F8.4,' SLOPE=',F8.4,
2  ', WTOT=',F8.4)
300  CONTINUE
      RETURN
      END

```

```

C
C
C
C
      SUBROUTINE EDGE(NSTT,NSTP,LTIME,WAV1,WAV2)
C      EDGE FIXING ROUTINE CALLED BY FILTR
      DIMENSION WAV1(LTIME),WAV2(LTIME)
C
      DO 10 K=1,NSTT
10     WAV2(K)=WAV1(K)
      DO 20 K=NSTP,LTIME
20     WAV2(K)=WAV1(K)
      RETURN
      END

```



## Appendix L

TSD.FOR A program to compute various signal detection parameters given Hit and False Alarm Rates.

```

C
C F.TSD — A PROGRAM TO CALCULATE SEVERAL MEASURES OF
C SENSITIVITY AND BIAS FROM SIGNAL DETECTION
C THEORY GIVEN THE FOLLOWING:
C
C PROGRAMMER: DAVID STRAYER
C UNIVERSITY OF ILLINOIS
C
C
C NUMBER OF HITS
C NUMBER OF FALSE ALARMS
C
C THE INPUT CAN BE RAW SCORES, OR PROPORTIONS RELATIVE TO
C CATEGORY (E.G.,  $p(\text{HIT})+p(\text{MISS})=1.0$  AND  $p(\text{FA})+p(\text{CR})=1.0$ )
C
C INTEGER*1 IANS
C REAL HIT, FA, CR, MISS, APRIME, BPMPM, AG, E, PS, BETA, LNBETA,
+DPRIME, Z1, Z2, TEST1, TEST2, C0, C1, C2, D1, D2, D3, K,
+T1, T2, T3, T4, B1, BPRIME
C
1 WRITE(3,*) 'Enter HIT rate'
  READ(0,*) HIT
  IF(HIT.GT.1.0) THEN
    WRITE(3,*) 'Enter MISS Rate'
    READ(0,*) MISS
    HIT=HIT/(HIT+MISS)
  END IF
  MISS=1.0-HIT
C
  WRITE(3,*) 'Enter False Alarm (FA) rate'
  READ(0,*) FA
  IF(FA.GT.1.0) THEN
    WRITE(3,*) 'Enter Correct Rejection (CR) rate'
    READ(0,*) CR
    FA=FA/(CR+FA)
  END IF
  CR=1.0-FA
  WRITE(3,*) '
  WRITE(3,*) '
  WRITE(3,*) '
  WRITE(3,98) HIT, MISS
98 FORMAT(' Target presented : ',F3.2,' : ',F3.2,' : ')
  WRITE(3,*) ' : HIT : MISS : '
  WRITE(3,*) ' :.....: '
  WRITE(3,99) FA, CR
99 FORMAT(' Distractor presented : ',F3.2,' : ',F3.2,' : ')
  WRITE(3,*) ' : FA : CR : '
  WRITE(3,*) ' :.....: '
  WRITE(3,*) '
  WRITE(3,*) ' Are ALL parameters correct? (Y/N)'
  READ(0,2) IANS

```

```

2      FORMAT(A1)
      IF(IANS.NE.'Y'.AND.IANS.NE.'y') GOTO 1
C
C      FUDGE FACTORS TO PREVENT DIVISION BY ZERO
C
      IF(HIT.EQ.1.0) THEN
          HIT=.9999
          MISS=1-HIT
      END IF
C
      IF(HIT.EQ.0.0) THEN
          HIT=.0001
          MISS=1-HIT
      END IF
C
      IF(FA.EQ.1.0) THEN
          FA=.9999
          CR=1-FA
      END IF
C
      IF(FA.EQ.0.0) THEN
          FA=.0001
          CR=1-FA
      END IF
C
C      FORMULA FOR CALCULATING A~ (A NONPARAMETRIC MEASURE OF EFFICIENCY)
C
C      FROM WICKENS(1984), (P. 502) SEE ALSO POLLACK AND NORMAN (1964)
C      RECOMMENDED BY PARASURAMAN AND DAVIES AS THE BEST MEASURE OF
C      SENSITIVITY
C
C       $A^{\sim} = 1 - .25 * ((FA/HIT) + (1-HIT)/(1-FA))$ 
C
C      APRIME =  $1 - .25 * ((FA/HIT) + (1-HIT)/(1-FA))$ 
C
C      FORMULA FOR CALCULATING AG (ANOTHER NONPARAMETRIC AREA MEASURE)
C
C      FROM CRAIG, 1979 SEE ALSO GREEN AND SEWTS (1966)
C       $A_g = (HIT + (1-FA))/2$ 
C
C       $AG = (HIT + (1-FA))/2.0$ 
C
C      FORMULA FOR CALCULATING E (YET ANOTHER NONPARAMETRIC AREA MEASURE)
C
C      FROM CRAIG (1979) SEE ALSO MCCORNACK (1961)
C
C       $E = 1 - ((1-HIT) / ((1-HIT) * PS + (FA * PS) - FA))$ 
C      WHERE PS IS SIGNAL PROBABILITY (BATCH QUALITY)
C      (THIS PROGRAM ASSUMES PS=.5)
C

```



```

C      B'=(HIT(1-HIT)-FA(1-FA))/(HIT(1-HIT)+FA(1-FA))
C
C      BPMPM=(HIT*(1.-HIT)-FA*(1.-FA))/(HIT*(1.-HIT)+FA*(1.-FA))
C
      WRITE(3,*) ' Sensitivity measures: ',
+Dprime      Aprime      Ag      E-
      WRITE(3,100) DPRIME,APRIME,AG,E
      WRITE(3,*) '
      WRITE(3,*) ' Response Bias Measures: ',
+Beta      LnBeta      Bprime      BPMPM
      WRITE(3,100) BETA, LNBETA, BPRIME, BPMPM
      WRITE(3,*) '
      STOP
100    FORMAT(25X,4(F10.6,4X))
      END

```

## Appendix M

Written instructions given to the subject at the onset of the experiment, which describe the experimental procedure.

The experiment which you are about to participate in has been designed to study aspects of the way we use memory. It is extremely important that you follow the instructions that you are about to read very closely. Even if it seems that you could perform the task better by adopting a strategy different from the one you will be instructed to use, please do only as you are instructed. Please read the instructions carefully, and if you have any questions, please ask them before beginning the experiment.

During the experiment you will work at an IBM PC. Stimuli will be displayed on the computer's screen, and you will make "yes" and "no" responses by pressing keys designated by the experimenter. There will be many trials, each of which will proceed as follows:

- 1) You will see on the screen two or four words. They will remain on the screen for three seconds. While the words are on the screen, commit them to memory by repeating them over and over to yourself.
- 2) The words will disappear. When they do, a series of numbers may appear on the screen. (On some trials this part will not happen, and we will skip to step 3.) Watch the first two numbers carefully, but do nothing. When the third number appears, compare it to the first number you saw. If it is the same, respond "yes"; if it is not the same as the first number press the "no" key. When the fourth number appears, compare it to the second number, responding "yes" or "no" as you did for the third number. Continue comparing every number that appears on the screen to the number two before it. I.E.: The fifth number will be compared to the third number, the sixth number will be compared to the fourth number, etc.. Respond "yes" or "no" as quickly and accurately as you can. This part of the experiment is difficult. Do not be discouraged if it takes you a little while to get good at it. Do not think about the words you saw at the start of the trial while you are doing this task.
- 3) An asterisk will appear on the screen. This signals you that it is time to remember the words that you saw at the start of the trial.
- 4) A word will appear on the screen. Respond by pressing the "yes" key if it is one of the words you saw at the start of the trial. Respond "no" if it is not. Respond as quickly as you can while maintaining accuracy. If you do not know if the word was presented at the beginning of this trial, think for a little while, and then guess if you still cannot remember. Try to always make a response.
- 5) There will be a short pause, and then a new trial will begin.

Things happen quickly in this experiment so please be alert. There will be a break midway through each session. The experimenter will have to use your computer to arrange things for the second half. This is your opportunity to rest. Begin again when you feel you are ready.

Before you leave, make sure that you and the experimenter agree upon when your next session is scheduled to take place.

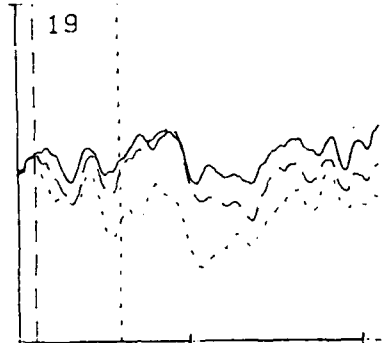
Thank you for your participation!



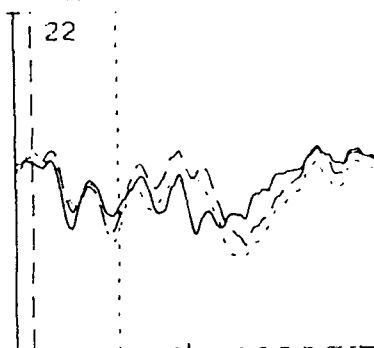
## Appendix N

Single subject scalp distribution ERPs collected using program PMSM.FLX. The solid line represents the Fz electrode, dash Cz, and dotted Pz. Ten single subject plots are included. Note that subjects 8 and 9 were rejected due to high error rates. Subject 9 also has a large amount of alpha activity. The baseline subtracted from each waveform is from -100 msec pre-S1 to S1. The number of trials included in each average is displayed in the upper left-hand portion of each plot. A description of the ERPS is included in the results section of the report.

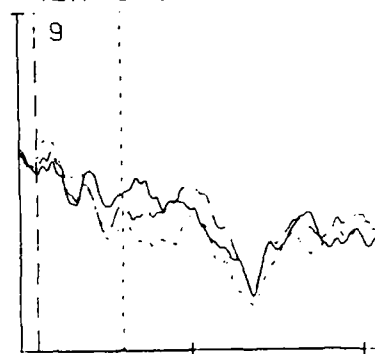
MEM 4 DE 0 TAR



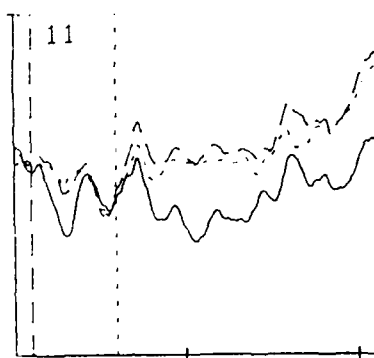
MEM 2 DE 4 NTAR



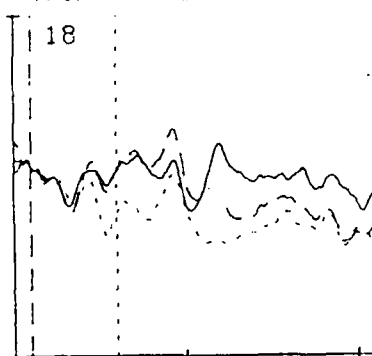
MEM 4 DE 15 NTAR



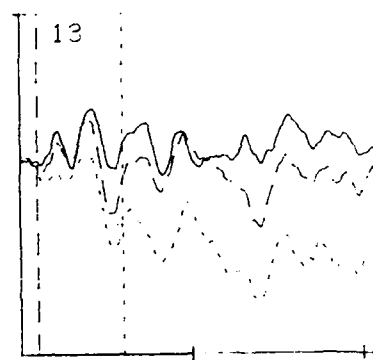
MEM 2 DE 15 TAR



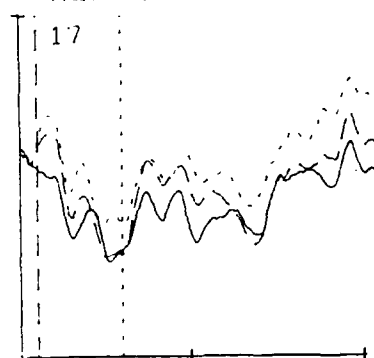
MEM 2 DE 0 NTAR



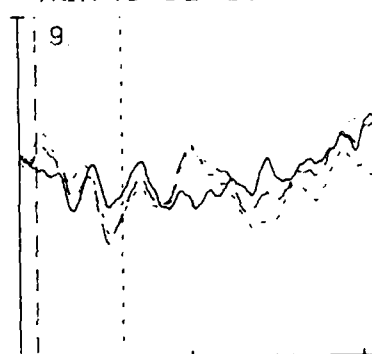
MEM 4 DE 4 NTAR



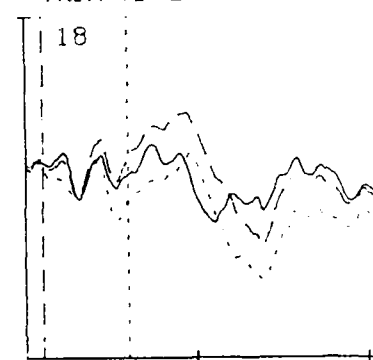
MEM 2 DE 4 TAR



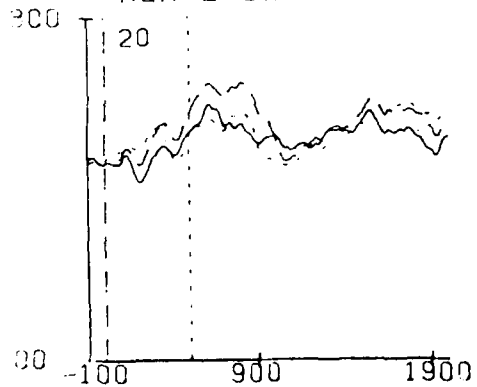
MEM 4 DE 15 TAR



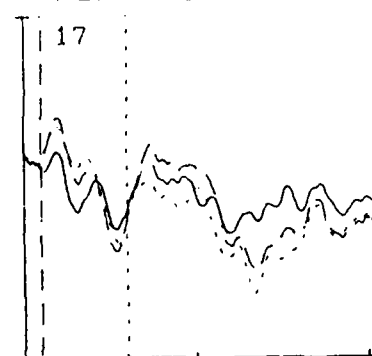
MEM 4 DE 0 NTAR



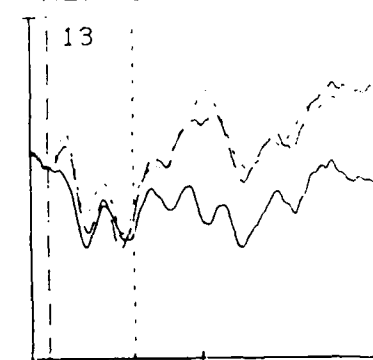
MEM 2 DE 0 TAR



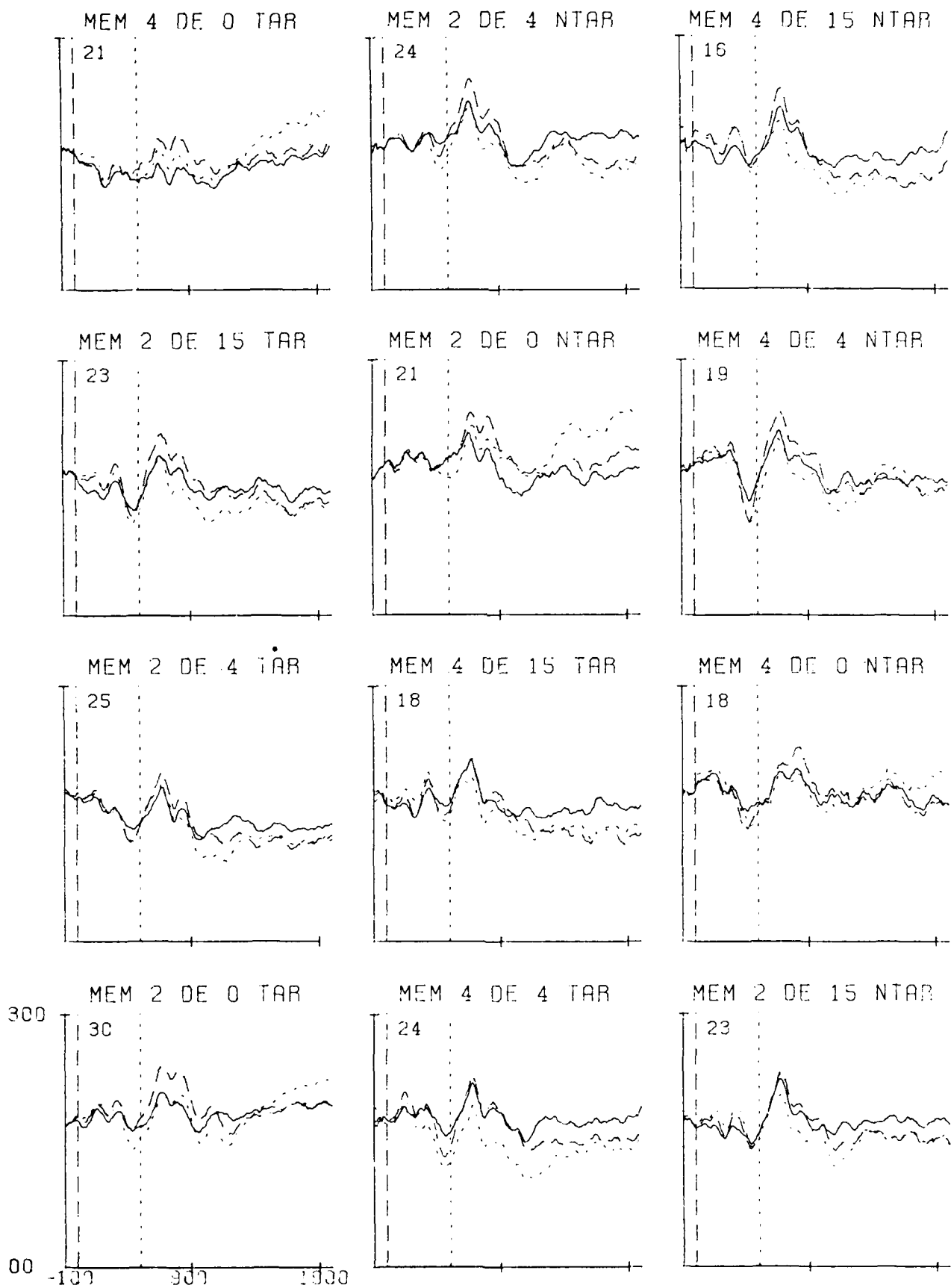
MEM 4 DE 4 TAR



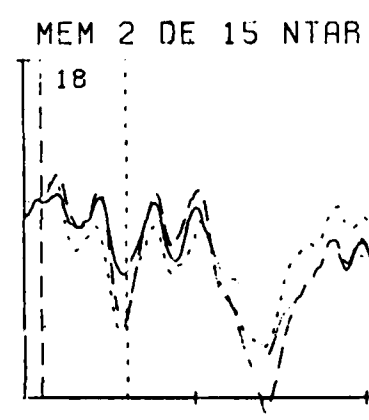
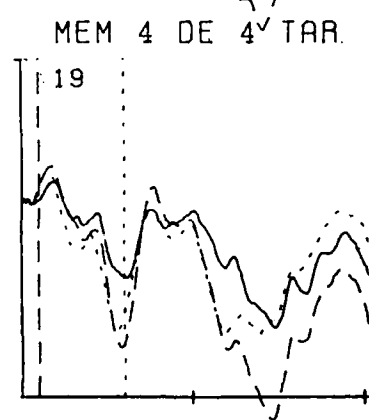
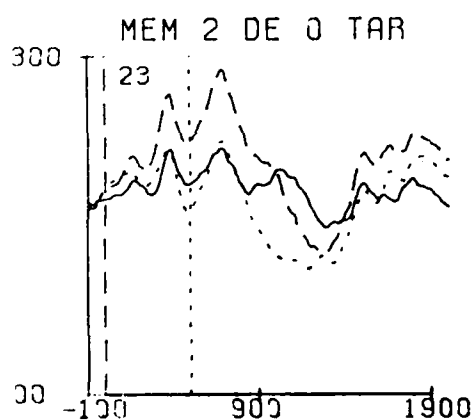
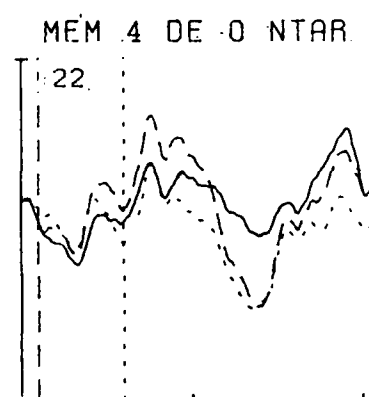
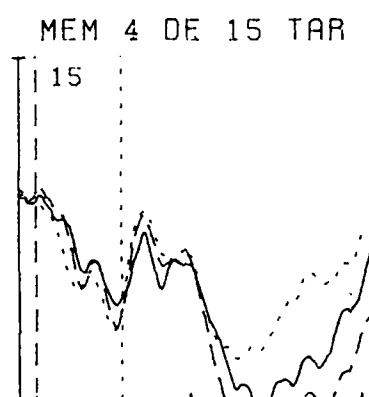
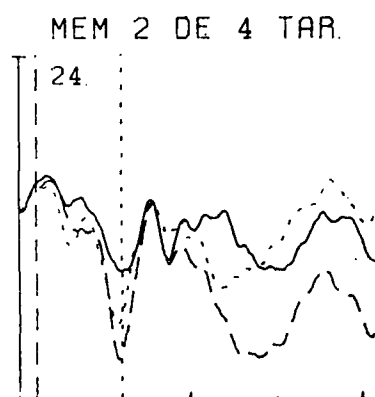
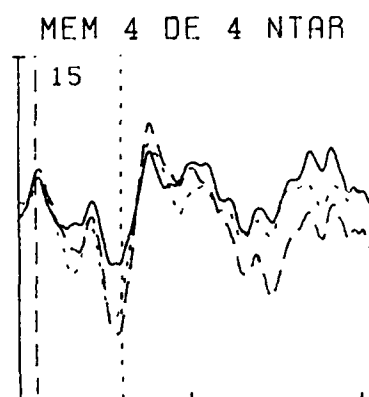
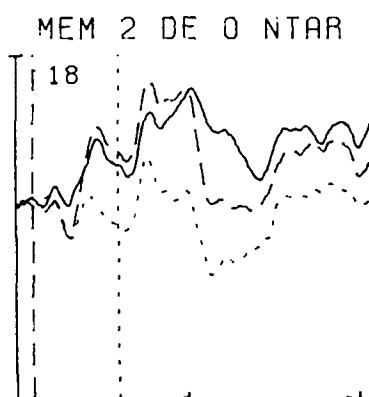
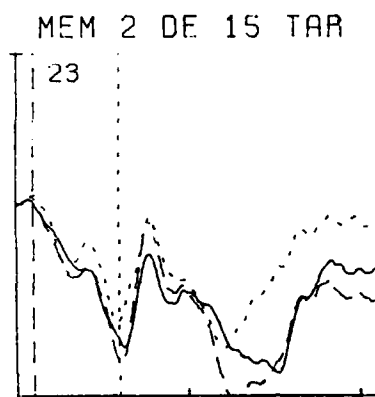
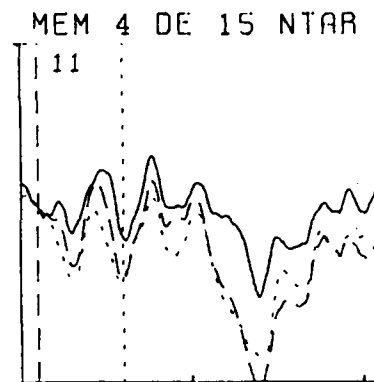
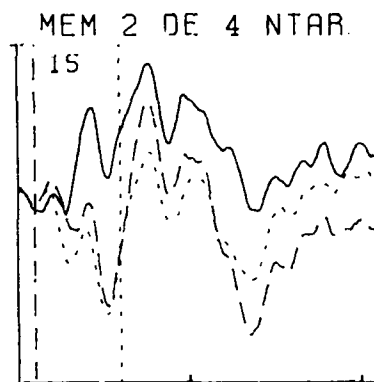
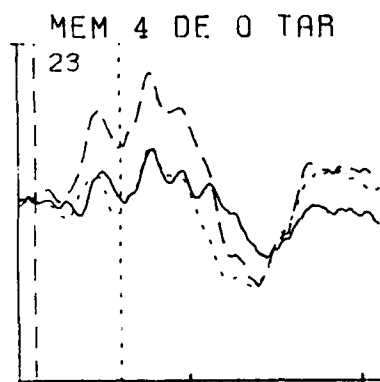
MEM 2 DE 15 NTAR



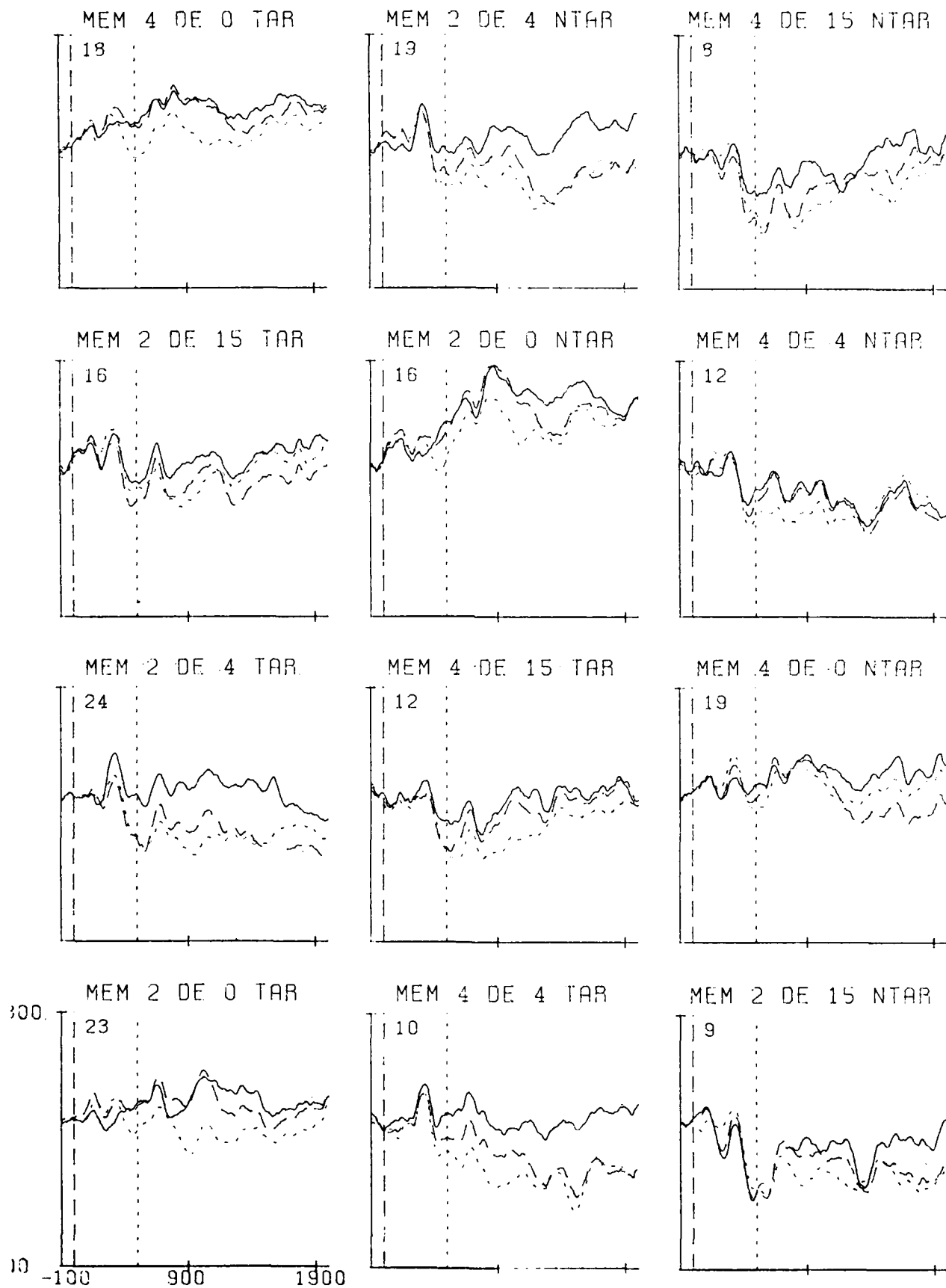
SUBJECTI



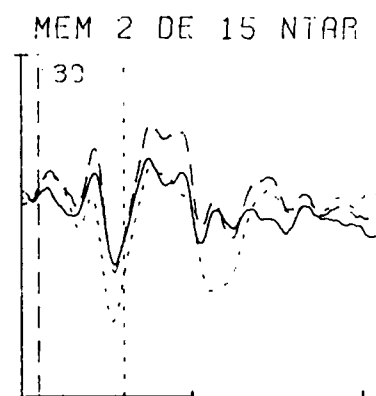
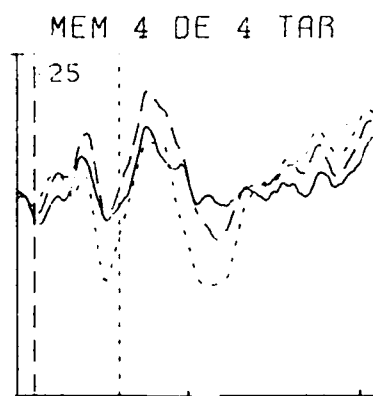
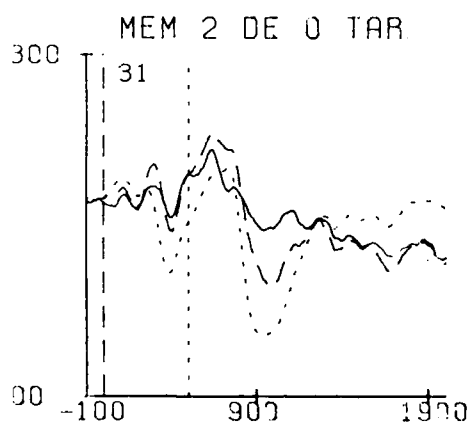
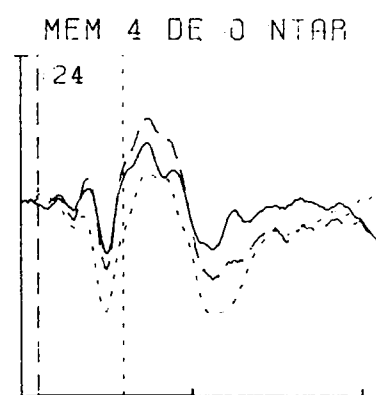
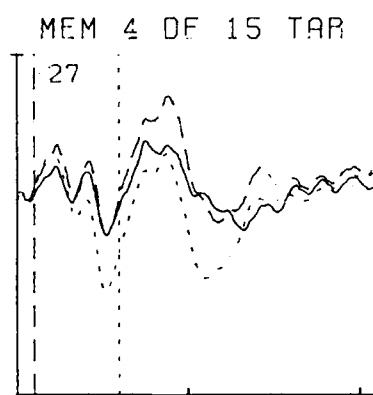
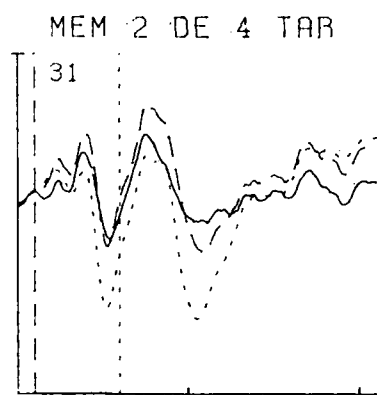
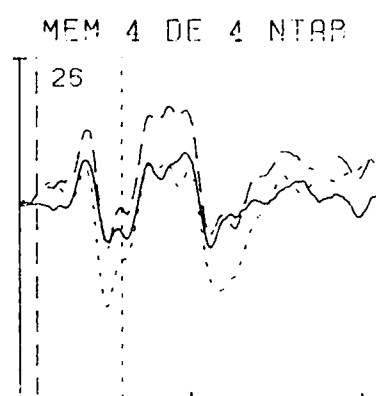
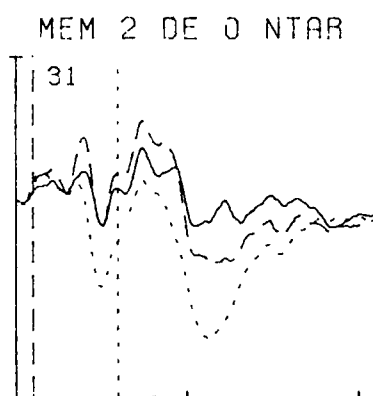
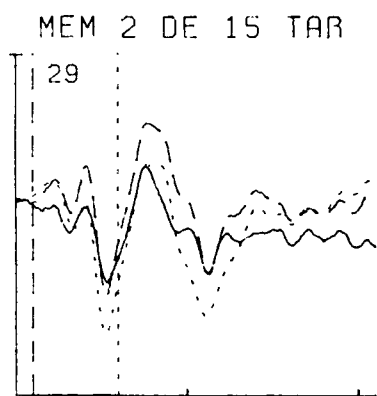
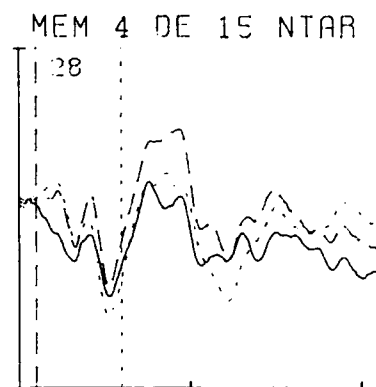
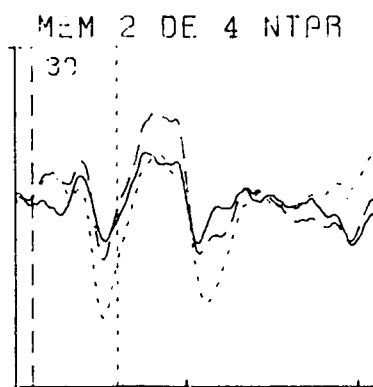
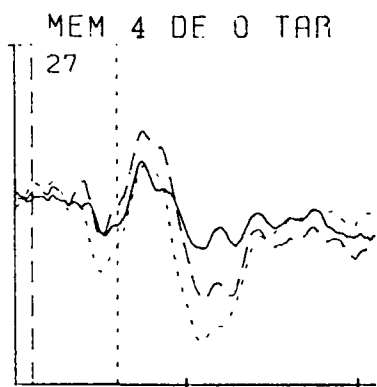
SUBJECT 2



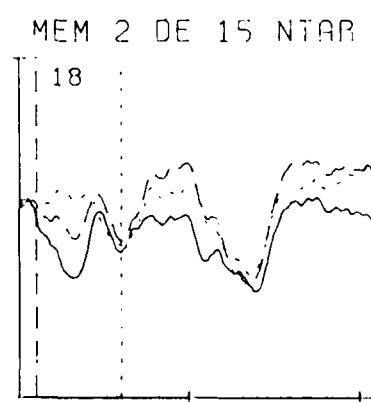
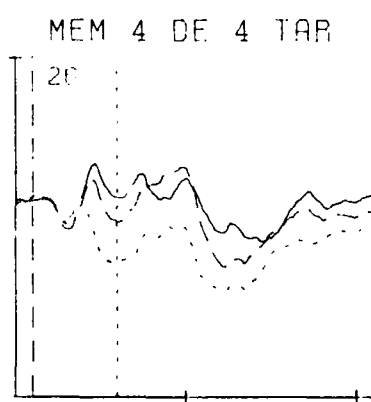
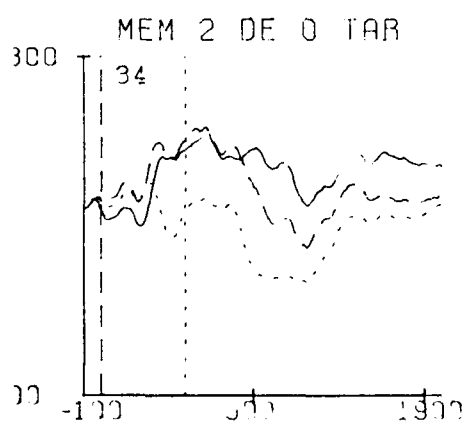
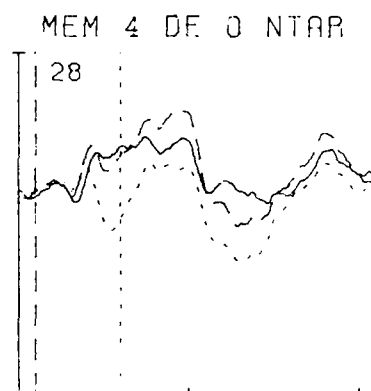
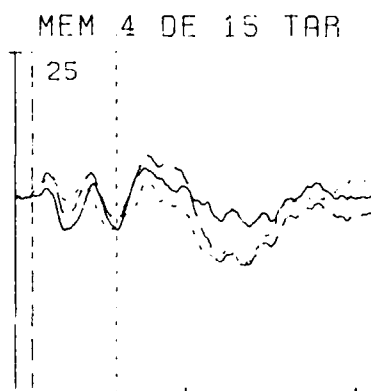
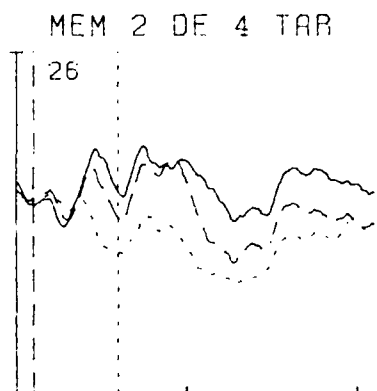
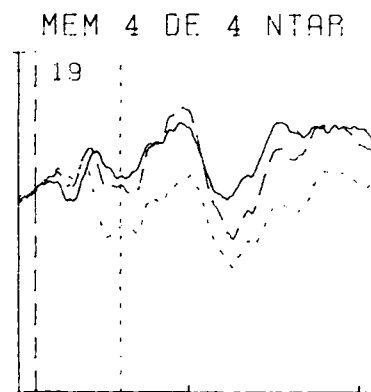
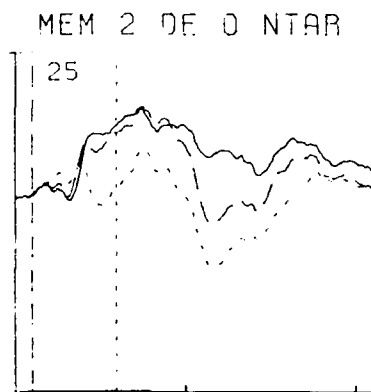
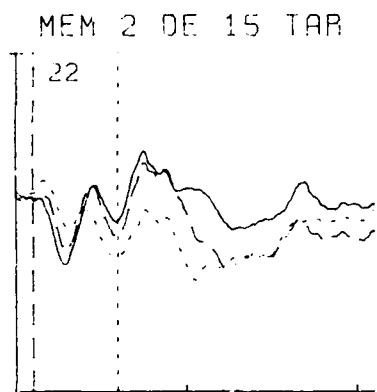
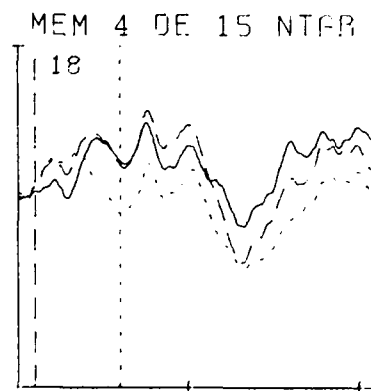
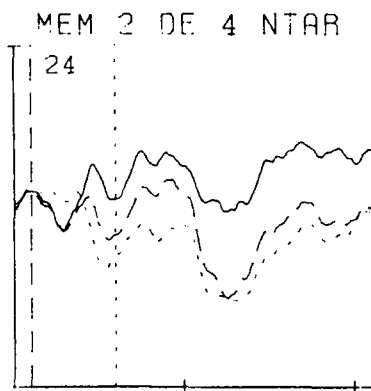
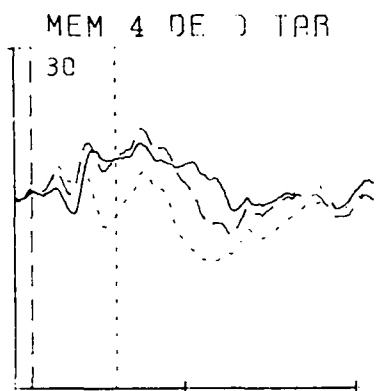
Subject 3



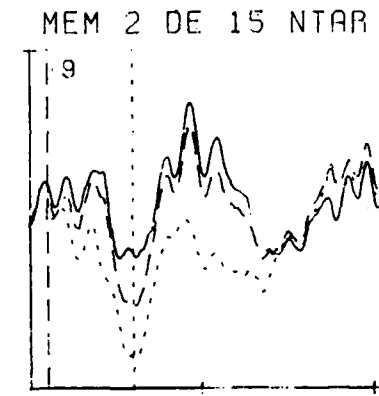
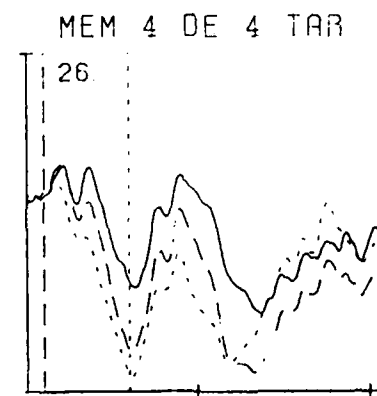
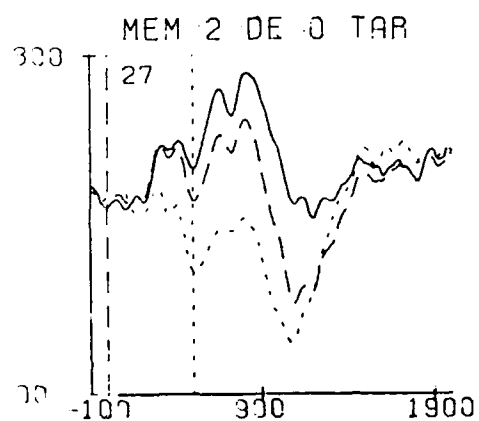
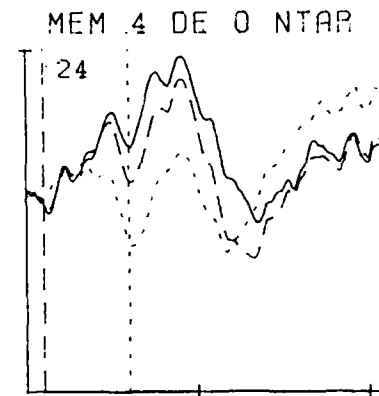
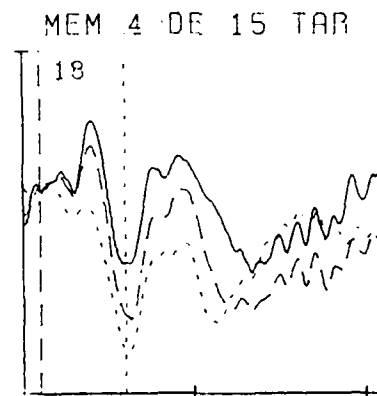
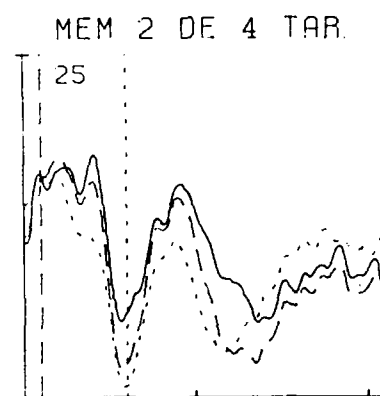
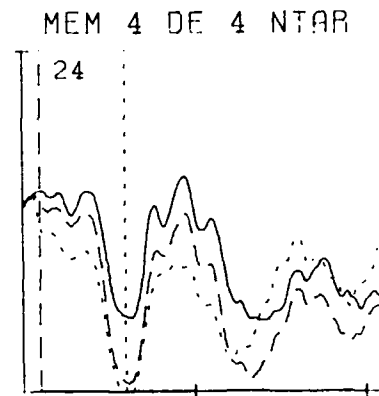
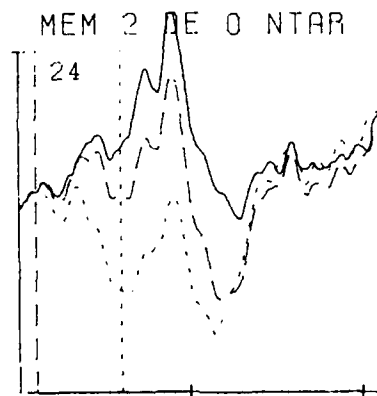
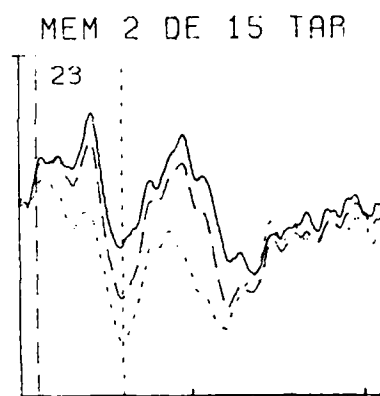
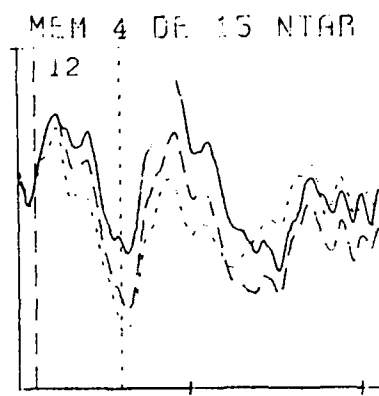
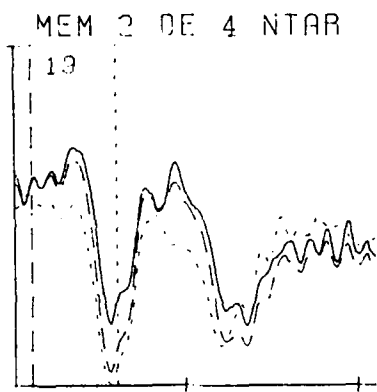
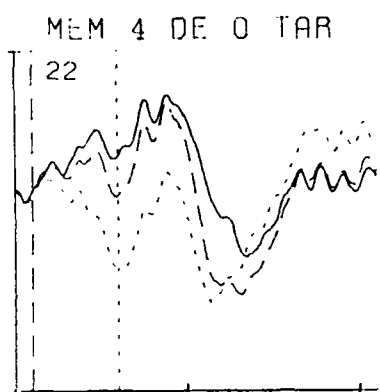
Subject 4



Subject 5

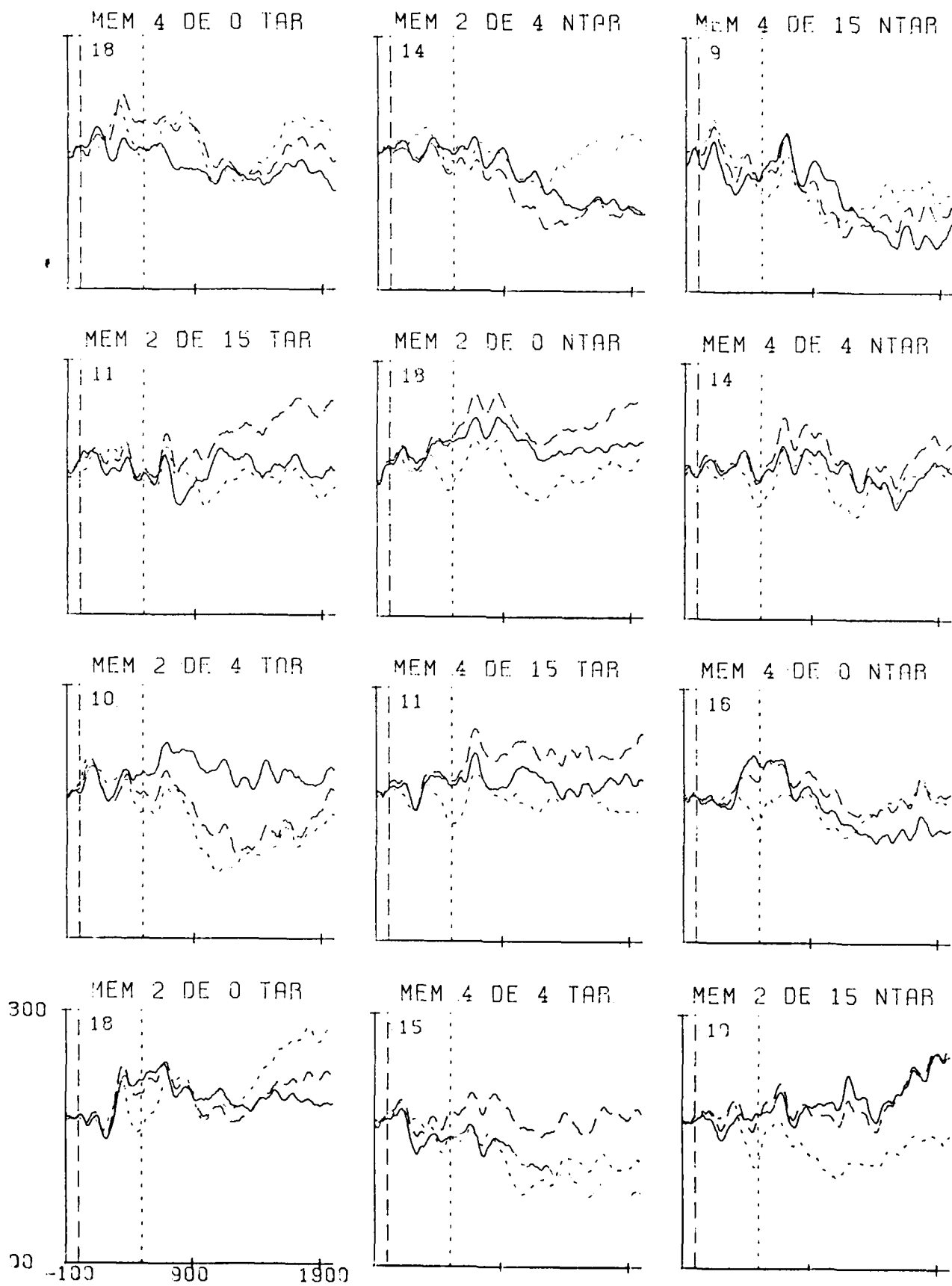


Subject 6

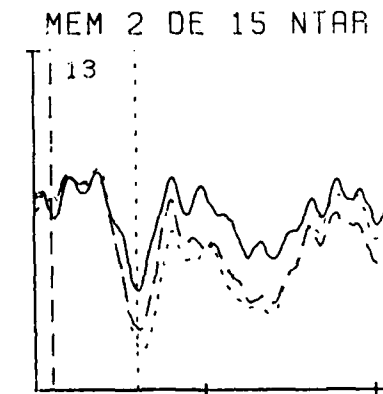
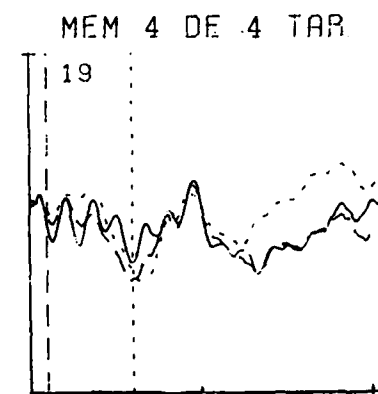
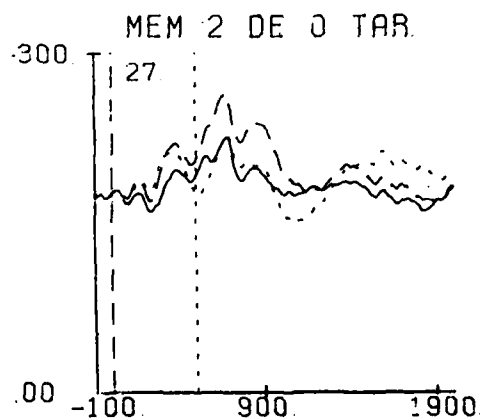
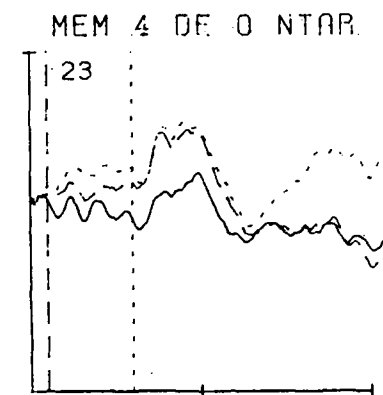
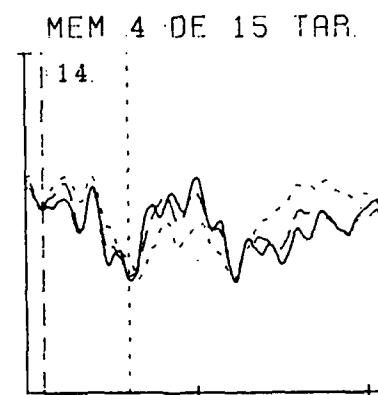
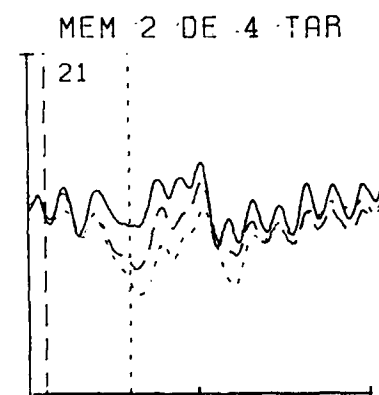
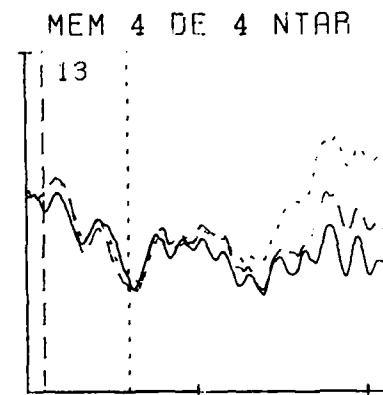
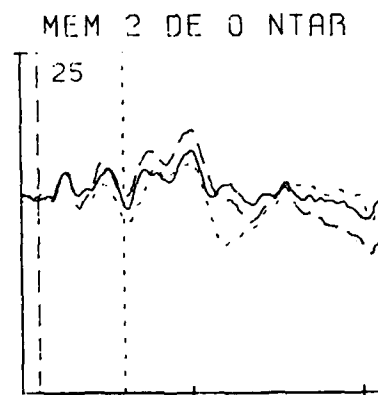
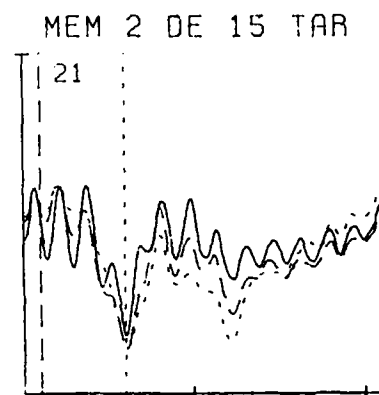
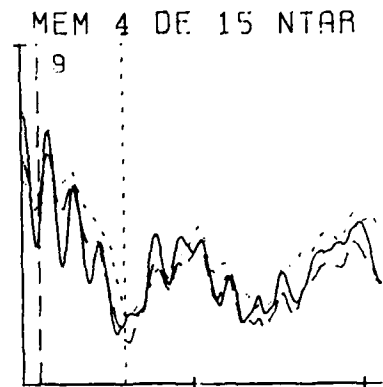
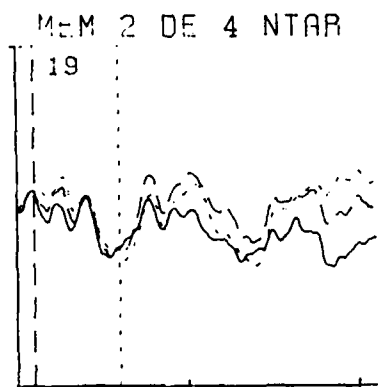
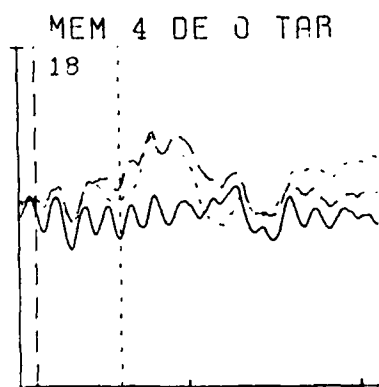


Subject 7





Subject B



Subject 9

